PCT-9 SERIES
SUBSYSTEMS

Including the ATC-8 and the ATC-16 interfaces.

MANUAL
VERSION
3.2

November 1, 1989.

2243 Agate Court · Simi Valley, CA 93065-1898 · (805) 584-2233

Copyright

Software Disclaimer

Catamount Corporation makes no warranties, either expressed or implied, with respect to this manual or with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Catamount Software is sold or licensed "as is". The entire risk as to its quality and performance is with the buyer. Should the programs prove defective following their purchase, the buyer assumes the entire cost of all necessary servicing, repair, or correction of the software and any incidental or consequential damages. In no event will Catamount Corporation be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software, even if Catamount has been advised of the possibility of such damages.

Trademarks

IBM-PC/XT/AT, PS/2 and PC-DOS are trademarks of International Business Machines Corporation. MS-DOS, MICROSOFT-C, QUICK-C and MICROSOFT MACRO ASSEMBLER are trademarks of Microsoft Corporation. Microstreamer is a trademark of Cipher Data Products, Inc. TURBO-C, TURBO PASCAL and TURBO ASSEMBLER are trademarks of Borland International. APPLE][is a trademark of Apple Computer Corporation.

Warning!

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with this instruction manual, may cause interference to radio communications. It has been tested and found to comply with the requirements for a class A computing device pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment.

Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures which may be required to correct the interference.

The user must use the cables supplied by the manufacturer to assure compliance with the requirements for class A computing equipment.

Table of Contents

Table of Contents

Table of Contents

The GPTR Software — NINE

The Catamount Controller Hardware — TEN

Error Messages — ELEVEN

Table of Contents

1/2" Tape Primer — APPENDIX A

Board Layout — APPENDIX B

About This Manual

This user's manual is intended to provide complete information on the use of the Catamount subsystems. It is assumed that the reader has a working knowledge of the PC-DOS or MS-DOS operating system and is familiar with file structures, the execution of commands and interpretation of format notations commonly used in working with these operating systems.

- First time users should read the Introduction and Installation chapters, Chapters 1-2, scan over the utility programs in general in Chapter 3, choose the software appropriate for their application based upon the information provided in Chapter 4, Software Selection, and proceed to the chapter discussing that software in depth.

- Chapters 5 and 6 discuss the Disk-to-Tape and Tape-to-Disk transfer utility program and the Disk-to-Tape backup and Tape-to-Disk restoration utility programs. These programs are self-contained and do not require comprehensive understanding of programming environments. Examples of commands using these programs are given at the end of each of these chapters.

- Chapters 7, 8, and 9 document the three support software environments, (device emulation, general purpose programming, and BASIC) provided for use by applications programs and/or system utilities. These chapters, and their corresponding environments, require comprehensive knowledge of the appropriate operating systems as well as the applications software and languages.

- Chapter 10 describes the details of each of the Catamount controller cards, and is provided for experienced programmers who wish to manipulate the controller directly.

- Chapter 11 covers the Catamount error codes, together with a brief explanation of each.

- Appendix A, "A 1/2. Tape Primer", covers the general subject of 1/2" magnetic tape recording, its applications and common usages and is recommended for all readers. This appendix also contains ASCII and EBCDIC charts that may prove useful in various applications.

- Appendix B describes the layout of the jumpers on each Catamount controller. Each diagram displays the default settings. This appendix also displays the jumper setup for each possible I/O address. For the MTC-16 there are no hardware jumpers. All hardware settings are controlled through software.

Notes

Chapter: ONE

Introduction

The Catamount ½ inch Magnetic Tape Subsystems are designed to provide access to, and use of, ½ inch magnetic tape data from the IBM PC/XT/AT and PS/2 computers and 100% compatibles. The Catamount subsystems support the IBM and ANSI compatible NRZI (Non-Return to Zero Invert) format at 800 bpi (Bits per inch), the PE (Phase encoded) data recording formats at 1600 and 3200 bpi, and the GCR (Group Coded Records) format at 6250 bpi depending on which subsystem have you purchased.

1.1 Hardware

The Standard Catamount subsystem consists of a tape drive, a desk-top enclosure, an interface card, data cable and software. The Catamount interface is a full function DMA interface allowing access to all features of the Catamount tape drive. The Catamount interface occupies one expansion slot on the computer's I/O channel.

A single data cable connects the interface card to either pair of P1 and P2 connectors on the rear of the tape drive.

Up to eight tape drives may be attached in a daisy-chain fashion to a single Catamount interface. Each tape drive attached in this fashion must be separately addressed and the termination resistor packs removed from all but the last drive in the chain.

1.2: Software

The Catamount Subsystems are supplied standard with five software packages. These software packages are used for data transfer to and from tape; backing up and restoring from floppy, hard disk or network drive; and custom program support.

1.2.1: DT

The **DT** program is used to copy data to and/or from magnetic tape to and/or from a DOS device, such as a disk drive or a printer. The DT program provides record blocking/deblocking, tab expansion and contraction, trailing blank removal or padding, carriage return and/or line feed insertion and removal, ASCII/EBCDIC translation and tape positioning for copied data.

1.2.2: TBACKUP and TRESTORE

The **TBACKUP** and **TRESTORE** programs are used to create tape backup copies of disk files. These commands function in a manner very similar to the DOS supplied BACKUP and RESTORE commands. Support is provided for tree structured directories, multiple disk drives, file selection by archive bit and date of last modification, global file names, multi-reel backups, file classification and file exclusion. Additional utilities are provided to scratch the backup tapes and to list directory information from the backup tapes.

1.2.3: TDS

The **TDS** (Tape Device Support) software provides DOS with additional devices used to transfer data to and from tape. One (or more, if additional tape drives are installed) block device(s) and one (or more) character device(s) are added to the system, with the additional block device(s) providing data as files that can be accessed through conventional operating system calls. ASCII/EBCDIC translation, record deblocking, trailing blank removal or padding, tab expansion and contraction, carriage return and line feed insertion and removal are selectable as options. These devices are controlled through DOS's IOCTL facility.

1.2.4: BPS

The **BPS** (BASIC Programming Support) software provides a simplified and fast method of accessing tape data through the BASIC language. String data is passed via the USR function to and from the tape drive. The BPS software provides optional ASCII/EBCDIC translation as well as record blocking and deblocking.

1.2.5 GPTR

The **GPTR** (General Purpose Tape Routines) is a library of machine language subroutines designed to be LINKed to other (usually high level language) programs, and provide access to standard tape drive functions. The GPTR subroutines are accessed by other programs when LINKed via the DOS LINK utility. The GPTR package is derived from the BPS software and provides similar features.

Notes

Chapter: TWO

Installation

Before attempting installation of this (or any other) hardware and/or software, make sure you have two known good backup copies of any files and/or software resident on your computer and any devices attached to it. Be sure that these backup copies are "write protected" *and know how to use these backup copies to recover any files that may be destroyed.* Although it is quite unlikely that the hardware installation will destroy any data stored on your computer, *the possibility nevertheless exists.*

2.1 System Installation

The Catamount subsystem is shipped in one, two or three boxes depending on the specific model ordered. The tape drive should be unpacked and installed in accordance with the instructions in the 'Tape Transport Technical Manual' (Appendix C of this manual). If more than one tape drive is to be installed on a single controller, pay special attention to the section on multi-transport operation.

Be sure to retain the tape drive box and packing material. In the unlikely event that the tape drive needs to be sent back to the factory, it must be returned in its original container.

Unpack the interface controller card, data cable, diskette(s), manual and test tape (test tape is only included with a full subsystem). The interface card should be installed in an unused expansion slot in your computer. The computer cover must first be removed, and a slot cover also removed from the slot to be used for the interface. Note that the rightmost slot is not fully compatible with the others on some models of the IBM computer (such as the IBM PC/XT), and if so, must not be used by the Catamount interface. Insert the interface card into the selected slot and screw the bracket attached to the interface card to the support provided for this purpose and replace the cover of the computer.

The data cable is attached between the **D-type** connector on the back of the interface card to the P1 and P2 connectors on the back of the tape drive. If the data cable is not already attached to the tape drive, the card edge connector marked J1 may be attached to either of the P1 card edge connectors on the back of the tape drive. Likewise, the J2 cable connector may be attached to either P2 card edge connector. The end of the cable connector marked to indicate pin 1 must be near the end of the card edge connector marked 1 (The other end of the connector is marked 50). ***Because there is no keying mechanism to insure proper installation, check and recheck this connection to verify correct installation.***

If the data cable is to be connected to a tape drive other than the Cipher units, consult the appropriate Tape Drive Installation and Operations manual for the location of the drive's interface edge connectors. These connectors may also be referenced as J1 and J2, J4 and J5, or P4 and P5 depending on the drive manufacturer.

The tape drive and computer may now be powered up. If your computer fails to complete its normal power up sequence, turn it off immediately and recheck the installation. If you are unable to locate the problem, call Catamount or your dealer for assistance.

Copy all of the files on the root directory of the distribution diskette to your work disk. If you are using a hard disk, it is a good idea to create a subdirectory such as C:\TAPE to contain these files to avoid confusion with other files already on your disk. Make sure the subdirectory is added to your system PATH.

At this point you may also want to copy the files from the GPTR subdirectory to your work disk if you intend to use the GPTR software.

2.2: Software Installation

The Catamount utility software and support packages (except for the GPTR) require that a configuration file, **TCONFIG.SYS**, be present in either the current directory of the default disk, or available in a disk and directory listed in the current search path as defined by the DOS PATH command.

This configuration file, **TCONFIG.SYS**, is edited and maintained via the TINSTALL program.

You may proceed to run the TINSTALL program, which includes a detailed description of the Catamount configuration process. This is accomplished by entering into the subdirectory that contains the Catamount software and typing the command TINSTALL. You should run through the INITIAL INSTALLATION to set up the controller card. Normally the default values should be used. When the TINSTALL program prompts to see if you wish to update the **TDS.DEV** file, you should respond with a **'Y'** followed by a carriage return and the PATH to which the **TDS.DEV** will be saved.

Now would also be an excellent time to modify (or create, if you do not already have) your AUTOEXEC.BAT file to include a PATH statement pointing to the disk drive and path containing your tape software and **TCONFIG.SYS** file so that your tape programs will be accessible by merely entering the program name. If you are not familiar with this file or command, consult your DOS manual.

2.3: Testing the Hardware

Review the 'Tape Transport Operations Manual' (Appendix C of this manual) and become familiar with loading and general operation of the tape transport.

To test the operation of the tape drive, load a scratch tape (with a write ring) onto the tape drive.

Before the tape drive is placed online, run the TPOS program supplied on the distribution diskette by entering:

```
TPOS  TAPE:/BOT
```

The TPOS program should respond with an error code 3, indicating that the tape drive is not ready. If an error number 16 is returned the card was either not installed correctly, the TINSTALL values were not set to the I/O address on the card, or there is a conflict of I/O addressing between the Catamount controller and another device.

Now place the tape drive online.

To test the tape drive, run the TBACKUP program with the following parameters:

TBACKUP *.* TAPE:/RO/I

This will execute a self test of the tape drive, then proceed to 'back up' the contents of the current directory to tape. The TBACKUP program should respond with :

```
TBACKUP Disk to Tape Backup Utility
Catamount Corporation Ver. x.x
(C) 19xx IDB Corp. All Rights Reserved

Rewinding . . .
Beginning Tape Test . . .
Beginning Backup . . .
C:\TAPE\ASCII.ASM
C:\TAPE\EBCDIC.ASM
.
.
.
C:\TAPE\TINSTALL.OVL
C:\TAPE\CONFIG.SYS
29 Files / 374135 Bytes Backed Up to Tape
Normal End of Backup
```

If the system "hangs" or fails to complete a normal end of backup, there could be either a problem with the installation or a compatibility problem. To further investigate the possibility of a compatibility problem, the TINSTALL program provides a detailed description of the resources utilized by the Catamount subsystem. In particular, you may wish to try different interrupt levels or no interrupts at all, or different I/O addresses. If you are unable to resolve the problem after reading the information displayed by the TINSTALL program and after checking Chapter 11, fill in the Technical Support Worksheet (IN PENCIL) on page 2-7, and contact Catamount or your dealer for assistance.

If the above test worked successfully fill out the Technical Support Worksheet on page 2-7 (IN PENCIL). Please look at page 2-7 now and become familiar with its contents.

If you have any future need to call technical support, you will need to have the Technical Support Worksheet filled out and ready when you call.

Also, run the TINSTALL program and select the 'Parameter Table Edit' option. When the parameter table is displayed, please make a screen dump of the table. If you ever need to reload the Catamount Software, this screen dump and the Technical Support Worksheet will be of use to you (or someone else down the line).

2.4: Installing the TDS Device Driver

The **TDS** *device driver is not needed to use the primary Catamount utilities such as TBACKUP, TRESTORE, DT, TPOS, or the GPTR tape routines.* The **TDS** device driver is available to give you *pseudo disk* control of the tape drive.

Before proceeding with this section, you may want to review the chapter in your DOS manual on "Configuring Your System". You should also have a bootable diskette available that is configured for your system as it exists before this installation to reboot your system in case the modifications you make prevent you from booting your system with the boot disk with which you are working.

The **TDS** device driver is used to define up to sixteen additional devices to the operating system, with each of the eight possible tape drives supported by the software defined twice, once as a character-type device and once as a block-type (disk) device. It should be noted that although these devices are defined to the operating system, they do not actually become useable until the **TDS.COM** program is executed.

The **TDS** device driver will add one to eight character devices, named TAPE0: through TAPE7: to your system. The fact that these devices are defined as such will prevent you from using any disk files named TAPE0 through TAPE7 on your system, just as you cannot name disk files CON or LPT1.

In addition to this, up to eight additional logical *disk drives* are also added to your system, assuming the next eight single letter disk drive identifiers available on your system. Thus, if the *highest* disk drive defined on your system before installing **TDS** was C:, the TDS software can define up to eight drives D: through K:.

If you are using the **TDS** software on a Local Area Network, note that the network may utilize Drives F: through Z:. If you have Hard drives from C: to E: the **TDS** software will not work, since there will not be any available disk drives free.

Anytime you change either the controller I/O address or the TDS configuration parameters in the **TCONFIG.SYS** file, you should use TINSTALL to create a new **TDS.DEV** file for inclusion in your system boot-up and reboot your system before using the TDS software.

Once the **TDS.DEV** file is ready for installation, it should be copied to the root directory of the boot device. Although it is not absolutely necessary for this file to be in the root directory, if you place it in another directory or on another disk the DEVICE= statement in the CONFIG.SYS file must be modified to include the disk and/or path name to the **TDS.DEV** device driver.

You may now add the command **DEVICE=TDS.DEV** to the CONFIG.SYS file in the root directory of your boot disk. If you do not already have a CONFIG.SYS file, the CONFIG.SYS file on the distribution diskette may be used, as it contains the **DEVICE=TDS.DEV** command. If you already have a CONFIG.SYS file, you may modify it by concatenating the CONFIG.SYS file on the distribution diskette with the CONFIG.SYS file in the root directory of your boot disk. This may be accomplished as follows:

```
C:\>RENAME  CONFIG.SYS  CONFIG.OLD
C:\>COPY  CONFIG.OLD+A:CONFIG.SYS  CONFIG.SYS
```

As an alternative, you may simply edit in the **DEVICE=TDS.DEV** command using the

text editor of your choice.

Whatever method you use, it is a good idea to keep the old version of CONFIG.SYS on your boot disk, preferably under the name CONFIG.OLD, until you have satisfactorily completed this installation.

If you have a LASTDRIVE= command in your CONFIG.SYS file, you will need to modify this specification to reflect the additional drives. You may also have to modify other software on your system to reflect the different disk drive designators if you include the **DEVICE=TDS.DEV** command before other **DEVICE=** commands that also define disk drives within the CONFIG.SYS file.

With the CONFIG.SYS file now updated, and the **TDS.DEV** file available to the operating system on boot up, you should now test your installation. *Make sure you have a bootable diskette available to reboot your system if the installation is not successful.*

Perform a soft boot in the usual fashion by holding down the **Ctrl** key and the **Alt** key and striking the **Del** key. If your system fails to come up, boot off your bootable floppy disk (*you may have to power the system down to do this*), restore your original CONFIG.SYS file from either a backup disk or other device or by erasing the new CONFIG.SYS file and renaming the CONFIG.OLD file to be CONFIG.SYS. Recheck these instructions and reread the section on Configuring Your System in the DOS manual. If after reattempting the installation your system still will not boot, contact Catamount or your dealer for assistance.

To verify that you have correctly installed the TDS device driver, you may attempt to do a directory to one of your *new* logical disk drives. Again assuming that the highest drive in your system before installing the TDS device driver was drive C:, try the following:

DIR D:

You should immediately receive an error or general device failure reading drive D: and a prompt asking you whether you wish to Abort, Ignore or Retry. If the installation was not successful, you will receive an *invalid drive specification* or similar error.

Likewise, you should also receive a general device failure or error reading device TAPE0: when you attempt a command such as:

COPY TAPE0: CON:

If the installation was not successful, you will receive a *file not found* error.

If you wish to proceed further with the TDS software, please refer to the chapter of this manual on TDS.

2.5: Technical Support Worksheet

Please Fill out this information. If you need to call Catamount For technical Assistance, you will need this information for faster service. **Please use pencil**, and if you change computers or any configuration information, make the necessary corrections to this page.

Date Purchased:	List of all controller cards in the computer, including the type of video interface and Disk interface.
Type Of Computer:	
Serial Number Of Controller Card:	
Serial Number Of Tape Drive:	
The values below are available in the TINSTALL configuration table:	
Base I/O Address:	
Interrupt Level:	
DMA Channel:	
Buffering Method:	

If you have a problem, please check Chapter 11 for a possible solution. If you cannot resolve the problem yourself, call Catamount Corporation Technical Support:

Phone: 805/584-2233
Hours: 8:00am to 5:00pm , Pacific Standard Time.
Days: Monday through Friday

Also have the following information ready:

- Steps performed before the problem occured.
- Error messages and symptoms that appeared.
- List of recent changes to your computer.
- List of recent additions to your computer software. (Any new TSR programs or Device Drivers.)

Notes

Chapter: THREE

Utility Programs

Several utility programs are provided with the Catamount controller. These programs, DT, TBACKUP, TRESTORE, TDIR, SCRATCH, TPOS, TDUMP, FREE and IOCTL, are executed as commands and are designed to be compatible with DOS and function in a manner similar to standard DOS commands. (DT, TBACKUP, TRESTORE, TDIR and IOCTL are documented elsewhere in this manual).

3.1 General Information

As with other DOS commands, these receive parameters from the command line selecting the particular options and files to be used in the execution of the utility program.

In general, the rules listed in the DOS manual for typing DOS commands apply. For example, the DOS delimiters space, comma, semicolon, equal sign and the tab key are all valid delimiters for the tape software.

Because of this DOS-compatible approach, the special DOS functions such as piping, redirection, filtering and batching all work with these utilities just as they work with the DOS utilities.

3.1: (...continued)

You may, for example, place a valid redirection command in between any two parameters on the command line of a tape utility program, separated from the tape parameters by valid DOS delimiters, anywhere where it is valid to have a delimiter in the tape utilities.

Also, anywhere a filename appears in a tape utility command, (except as explicitly noted in the TBACKUP and TRESTORE command sub-parameters) you may put a DOS file specification consisting of a valid character device name (optionally followed by a colon) or a disk, path, filename and/or extension. In cases where multiple files are being processed, such as with the TBACKUP, TRESTORE and TDIR utilities, global file name characters, "*" and "?" are permitted in accordance with the same rules as are used by DOS.

All Catamount programs return exit codes upon completion. This code may be tested with the '**IF ERRORLEVEL** *n*' facility from within DOS batch command streams. If the program completes without encountering errors, the exit code returned is zero. If an error is encountered, the error code printed on the standard output device will be the exit code returned.

3.2: The TAPE[n]: Specification

Most of the tape utility programs require that a specification of the tape drive appear in the command line. In these cases, you may use TAPE: to indicate logical tape drive zero, or TAPE0:. The other seven logical tape drives are indicated by the designations TAPE1: through TAPE7:. For these specifications the colon after the TAPE[*n*] is required.

The drive tape specification appearing in the DT, TBACKUP, TRESTORE, TDIR, TPOS and SCRATCH utilities is not a file specification per se. This is an identifier unique to the Catamount software and can be considered a syntactical structure. These specifications are easily confused with the TDS units TAPE0: through TAPE7:, and in fact reference the same logical tape drives. (This similarity is intentional). The TDS units, however, are actual operating system devices. It may be important in certain instances to understand when you are working with the TDS units and when you are simply selecting a tape drive for use with a Catamount utility program.

Also along this vein, it is important to understand that there is no communication or sharing of resources between the various tape utilities, except for the fact that they use a common configuration file and common tape drives. You must not expect, for example, for data in the TDS tape buffer to be properly flushed by writing a file mark with the TPOS utility. In this example, you must use the TDS command to write a file mark if you expect existing buffers to be respected.

3.3: The SCRATCH Utility

The SCRATCH utility is a general purpose program used to logically "scratch" a tape by writing two consecutive file marks at the beginning of the tape. Two consecutive file marks are traditionally used to mark the logical end of tape. Thus, this program marks the logical end of tape at the physical beginning of tape, indicating an "empty" tape.

The SCRATCH program may also be used to perform a "security erase", erasing data appearing after the logical end of tape yet before the physical end of tape. This assures that sensitive data left over from a prior usage of the tape does not appear on the unused portions of the tape after the tape has been reused.

This program is primarily provided to scratch tapes for users of the TBACKUP program so that automatic appending of tapes will not take place. The TBACKUP program recognizes tapes beginning with two file marks as empty tapes and proceeds to label them and use them as scratch tapes from the beginning.

SCRATCHing a tape for use with Catamount utilities other than TBACKUP is usually not necessary, as these utilities are able to work with various tape labels and protocols and therefore proceed to write from whatever position the tape is in unless they are instructed to reposition the tape, and it is not necessary to SCRATCH a tape prior to use, the tape need only be rewound.

The SCRATCH utility is helpful in other instances however, when the protocol defined by the user requires a scratch tape. An example of this is a .BAT file, for instance which transfers the days transactions to tape. This file could always position at end of tape using an /EOT option in a utility program and backspace then over one of the two file marks with a skipping option or a /A (append) option and proceed to append the days transactions to the ones from prior days already on tape. In this example, when the transactions are collected and processed, say at the end of the week, the tape can be scratched and reused.

3.3.1: SCRATCH Command Syntax

The SCRATCH program command syntax is as follows:

SCRATCH TAPE[*n*]:[/UNL][/ALL][/REST]

The *n* optionally appearing in the TAPE[*n*] specification is used to select which logical tape drive is to be used in the scratch operation if a tape drive other than drive 0 is to be used. To use tape drive 0 (the normal case), simply specify TAPE: or TAPE0:.

Each parameter begins with a "/" character. As many parameters as are appropriate for your application may be specified. When more than one parameter is to be specified, there must be no spaces or other characters between the parameters. Additionally, there may not be any characters separating the colon following the TAPE[*n*]: specification and subsequent parameters.

3.3: (...continued)

One or more spaces (or other valid DOS delimiter) are used to separate the command name SCRATCH from the TAPE[*n*]: specification.

/UNL specifies that the tape is unloaded after the completion of the scratch.

/ALL specifies that a security erase is to be performed upon the entire physical tape after writing two file marks to the beginning of the tape. To prevent a runaway tape condition if this tape is read past the initial file marks, two additional file marks are written after the physical end of tape reflector.

/REST functions in a manner similar to the /ALL option, but, the tape is not rewound prior to commencing with the writing of two file marks and the security erase. This allows a security erase to be performed after writing data to tape, assuming that the tape is not repositioned or rewound after writing the last data to the tape.

3.3.2: Examples Using SCRATCH

To logically scratch the tape, enter the following command:

SCRATCH TAPE:

To scratch a tape and physically erase all information on the tape from the physical beginning-of-tape reflector to beyond the physical end-of-tape reflector (performing a security erase), and unload the tape when done:

SCRATCH TAPE:/ALL/UNL

3.4: The TPOS Utility

The tape positioning utility, TPOS, may be used to preposition a tape. This is frequently neccessary when using the DT program to copy to or from tape, or to perform a positioning function that is not provided for by the particular utility program that follows.

3.4.1: TPOS Command Syntax

The TPOS program command syntax is as follows:

TPOS TAPE[*n*]:[*parameter...*]

The *n* optionally appearing in the TAPE[*n*] specification is used to select which logical tape drive is to be used in the positioning if a tape drive other than drive 0 is to be used. To use tape drive 0 (the normal case), simply specify TAPE: or TAPE0:.

Each parameter begins with a "/" character. As many parameters as are appropriate for your application may be specified. When more than one parameter is to be specified, there must be no spaces or other characters between the parameters. Additionally, there may not be any characters separating the colon following the TAPE[*n*]: specification and subsequent parameters.

One or more spaces (or other valid DOS delimiter) are used to separate the command name TPOS from the TAPE[*n*]: specification.

The parameters that may follow the TAPE[*n*]: specification are:

/BOT is used to rewind the tape to the load point.

/EOT is used to position the tape to the logical end of tape. The logical end of tape is indicated by two consecutive file marks. This option causes the tape to be spaced forward until two consecutive file marks are found, and then backspaced over one of the file marks. This leaves the tape positioned so that additional files may be added to the tape, or so that the tape may be backspaced (using the /RSF:*n* parameter) so that the last file on the tape may be appended.

/SB:*n* specifies that the tape is to be positioned by spacing the tape over *n* physical tape blocks in the forward direction. If a file mark is encountered while performing this spacing, the operation is aborted and the TPOS program is terminated.

/SF:*n* specifies that the tape is to be positioned by spacing the tape over *n* file marks in the forward direction. If two consecutive file marks are encountered, indicating the logical end of tape, the operation is aborted and the TPOS program is terminated.

/RSB:*n* specifies that the tape is to be positioned by spacing the tape over *n* physical tape blocks in the reverse direction. If a file mark or the physical beginning of tape is encountered while performing this spacing, the operation is aborted and the TPOS program is terminated.

/RSF:*n* specifies that the tape is to be positioned by spacing the tape over *n* file marks in the reverse direction. If the beginning of tape is encountered while performing this spacing and there are more than one additional file marks to be spaced over, the operation is aborted and the TPOS program is terminated.

/WFM is used to write two consecutive file marks to tape and then position the tape by backspacing over one of the file marks. This leaves the tape in a position such that additional files may be appended to the tape without further repositioning, or the preceeding file may be appended by then backspacing the tape over one file mark.

/UNL is used to initiate a rewind and unloading of the tape. For tape drives not equipped with a single line rewind and unload control, this command will cause the tape drive to go offline.

The operations performed by the TPOS program are performed by interpreting the parameters in a left-to-right fashion, one at a time, until either all parameters have been processed or an error condition is encountered.

3.4.2: Examples Using TPOS

To remove the last block of data from a reel of tape, the following command could be used:

```
TPOS  TAPE:/EOT/RSF:1/RSB:1/WFM
```

This command positions to after the file mark following the last block of data on the tape, backspaces over it, then backspaces over the data block preceeding that and writes two file marks over that block, indicating a new logical end of tape.

To position the tape after a set of labels at the beginning of a tape, use the following:

```
TPOS  TAPE:/BOT/SF:1
```

3.5: The FREE Utility

The BPS and TDS programming environments function by installing service routines in memory that are accessed through dedicated interrupt vectors to perform tape I/O and control functions. These programs utilize the DOS TSR (Terminate and Stay Resident) system function to remain resident in memory until explicitly dismissed, much as the resident portion of the DOS PRINT program remains in memory after a PRINT command is executed.

The FREE utility is used to remove TDS and/or BPS modules from memory, making the memory previously occupied once again available to the operating system. This is frequently helpful, as spread sheet and word processing programs often require as much memory as possible to function efficiently and the TDS and BPS routines can occupy very large amounts of memory when installed via execution of the **TDS.COM** or **BPS.COM** programs, particularly when large buffers or multiple tape drives are requested. The TDS and/or BPS modules may be reinstalled if needed for subsequent processing.

The FREE program has no parameters. To execute it, simply enter the command:

```
FREE
```

The FREE program will search a linked list for TDS and BPS modules to be dismissed and removes all modules found.

It should be noted that all the memory made available through the FREE program may not be available immediately to the operating system. In fact if other modules, such as PRINT, have been installed on top of the TDS or BPS module(s), it is unlikely that the operating system will ever recover use of the freed memory, as the memory will have become fragmented by having a block of occupied memory in the middle of two sections of free memory. Also, the operating system seems to be unable to combine more than one block of free memory at a time with the current program area, so if more than one BPS or TDS image is loaded when the FREE program is executed, additional space will appear each time any program is terminated until all the free space is recovered, as the operating system scrounges space upon completion of programs. This scrounging of free space occurs with the termination of any program, not just the FREE program.

3.6: The TDUMP Utility

The TDUMP utility is used to create a printout or listing of the contents or format of a tape. TDUMP is useful for analyzing and debugging tape data transfers.

Data may be displayed in hexadecimal, ASCII and EBCDIC form, or omitted from the listing entirely to provide a format-only summary.

Tape data may be partitioned into logical records, simplifying the location of fields within blocked records.

Additional options are provided to position the tape, suppress unwanted data listing, alter the page size of the listing and select special format modes.

3.6.1: TDUMP Command Syntax

The command syntax for the TDUMP program is as follows:

TDUMP TAPE[*n*]:[[*parameter*]...] [*dev:*][[*parameter*]...]

The dev: specification, if used, may specify any valid DOS device or file, including disk drive, path and file information, upon which the output listing is to be written. If no output device is specified, the listing is sent to the standard output device, normally CON:.

The *n* optionally appearing in the TAPE[*n*]: specification is used to select which tape drive is to be used in the dump if a tape drive other than drive 0 is to be used. To use tape drive 0 (the normal case), simply specify TAPE: or TAPE0:.

Fifteen different parameters are defined that may follow the TAPE[*n*]: specification in the command line, providing for positioning of the tape and selection of information to be displayed, as follows:

/ANSI indicates that the tape is an IBM and/or ANSI "Standard Labeled" tape, that labels are to be printed separately and that the file count is to count data files, not file marks. If this option is selected, labels must be 80 bytes long and the tape must begin with a VOL1 record. If the VOL1 record is in ASCII, EBCDIC data will not be displayed with the dump (unless overridden by the /EBCDIC parameter defined below). Likewise, if the VOL1 record is in EBCDIC, ASCII data will not be displayed. When this option is specified, the value found in the logical record length field of the HDR2 record will be used as the record length if no record length is specified in the command line. Selecting this option will rewind the tape to the physical beginning of tape if it is not already positioned there. This parameter improves the readability and applicability of the dump when used with ANSI or IBM standard labeled tapes.

/FILES:*n* specifies how many files are to be dumped. If the /ANSI option is selected, *n* specifies how many data files are to be dumped. If the /ANSI option is not specified, the contents of the tape through n file marks are dumped.

/ASCII specifies that the ASCII equivalent of tape data is to be displayed. This option allows the ASCII display to be presented on tapes that have EBCDIC labels when the /ANSI option is selected.

/NOASCII specifies that the ASCII equivalent of tape data is not to be displayed (Normally, the ASCII equivalent of tape data is displayed by the dump program, unless the /ANSI parameter is specified on EBCDIC tapes or the /FORMAT option is used).

/EBCDIC specifies that the EBCDIC equivalent of tape data is to be displayed. This option allows the EBCDIC display to be presented on tapes that have ASCII labels when the /ANSI option is selected.

/NOEBCDIC specifies that the EBCDIC equivalent of tape data is not to be displayed. (Normally, the EBCDIC equivalent of tape data is also displayed by the dump program, unless the /ANSI parameter is specified on ASCII tapes or the /FORMAT option is used).

/NOHEX specifies that the hexadecimal dump of tape data is to be suppressed. The hexadecimal dump of tape data is displayed unless either the /FORMAT or /NOHEX parameter is specified.

/FORMAT specifies that that only the tape format is to be dumped, indicating only the count of physical blocks of each length and the file marks. When the /ANSI parameter is specified along with the /FORMAT parameter, label information is displayed.

/R:*n* specifies the logical record length for records when being dumped. Specifying this parameter improves the readability of the dump in most cases, as offsets within records are more meaningful than offsets within blocks on tapes containing blocked records.

/BOT specifies that the tape is to be rewound prior to the tape dump operation.

/EOT specifies that the tape is to be positioned at the logical end of tape by spacing the tape forward until two consecutive file marks are found on the tape. The tape is then backspaced over the two ending file marks. This option is useful when checking ending labels, or when dumping the last or near last file, block or record on a tape.

/SB:*n* specifies that the tape is to be positioned by spacing the tape forward *n* physical blocks before the dump operation takes place. If a file mark is encountered while spacing the tape, an error condition will be reported.

/RSB:*n* specifies that the tape is to be positioned by spacing the tape *n* physical blocks in the reverse direction before the dump operation takes place. If a file mark or the physical beginning of tape is encountered while back spacing the tape, an error condition will be reported.

/SF:*n* specifies that the tape is to be positioned by spacing the tape forward *n* marks before the dump operation takes place. If the logical end-of-tape (two consecutive file marks) is encountered, an error condition will be reported.

/RSF:*n* specifies that the tape is to be positioned by spacing the tape *n* files in the reverse direction before the dump operation takes place. An error condition will occur if this operation attempts to position the tape prior to the physical beginning-of-tape. After successfully performing this positioning, the tape will be spaced forward over the last tape mark encountered, unless the tape is positioned at the physical beginning of tape.

/X:*filespec* allows the use of externally defined translation tables. These translation tables must be 512 bytes long, with the first 256 bytes containing the corresponding ASCII values to replace EBCDIC characters, and the next 256 bytes containing EBCDIC values to replace ASCII characters. The ASCII.ASM and EBCDIC.ASM source files used to create sample translation tables are included in this release. The external translation tables are specified by using a /X:*filespec* in the command line of the utility or installation program. The filespec contains a file name and optionally a disk drive designator, path, and or extent.

Two additional parameters are provided that affect the width and page length of the dump output. These parameters, if used, must follow the dev: specification, or a space or other valid DOS delimiter if the *dev:* specification is omitted:

/WIDTH:*n* specifies the number of tape characters per line to be displayed. Normally, the line length is equal to about five times this parameter plus ten bytes. If no value is specified, 8 characters of tape data are displayed per line. (This value is suitable for display on an 80-column display or printer. 80 Byte label records as listed by the /ANSI parameter, however, do not display well on all 80-column devices).

/LENGTH:*n* specifies the length (maximum number of lines per page) of the output file page. If a length of zero is specified, no headings are printed with the dump.

3.6.2: TDUMP General Operation

The positioning options are performed sequentially in a left-to-right fashion as they are encountered on the command line. Thus, if the file before the logical end of tape is to be dumped, the following command would be used:

TDUMP TAPE:/EOT/RSF:1

When using the positioning parameters, or when beginning the dump with the tape at a position other than at the beginning of the tape, the file, record and block counters displayed on the dump will be relative to the position of the tape when the actual dump of the tape format or data begins, not necessarily relative to either the beginning of tape or to the tape position prior to executing the TDUMP command.

ASCII and EBCDIC non-printing characters are replaced by periods "." in the ASCII and EBCDIC character displays on the dump. The dump itself uses the ASCII form-feed character (0CH) at the beginning of each page. This may print as a graphics-type character when sent to some displays.

The best way to familiarize yourself with the TDUMP utility is to try dumping a tape containing actual data of the type with which you work. Experiment with different format and display options.

3.6.3: Examples Using TDUMP

To dump the hexadecimal, ASCII and EBCDIC equivalents of tape data to the console:

TDUMP TAPE:

To dump the first file of a tape to the device PRN:, displaying 32 bytes of tape data per line, without the EBCDIC equivalent:

TDUMP TAPE:/NOEBCDIC PRN:/WIDTH:32

To create a file THIS.DMP containing the format dump of an ANSI labeled tape:

TDUMP TAPE:/FORMAT/ANSI THIS.DMP

Notes

Chapter: FOUR

Software Selection

4.1 Selecting Your Software

Your Catamount subsystem has now been installed. You must now determine which software package(s) is appropriate for your application.

The Catamount subsystem is primarily used for transferring data to and/or from tape for the purpose of exchanging data with another system. Exchanging data may or may not be difficult for any given application, depending upon the form of the data and the applicability of data types and meanings on one computer system compared with another. This discussion will assume that you are familiar with the data formats of both the source and destination computer system, languages and software. For a detailed discussion of the methods of recording 1/2" magnetic tape data, read "A 1/2" Tape Primer", Appendix A of this manual.

4.1: (...continued)

The TBACKUP and TRESTORE software is primarily provided as a convenience and is not used for normal data transfer between systems. These programs are provided to backup and restore data to and from a hard disk, as 1/2" tape provides a fast, efficient and economical method of storing archival and backup information when a 1/2" tape drive is already connected to a system.

The data exchange oriented software packages, TDS, BPS, GPTR and DT, are provided to assist data transfers in various applications.

4.2: The DT Program

The DT program is a general purpose program used to transfer data files from tape to disk and from disk to tape. If your application does not process massive amounts of data (i.e., sufficient disk space is available to provide intermediate storage of tape data), does not require extensive reformatting of data and is not severly impacted by the time required to process data twice (once from tape to disk, for example, and then from disk to the application software), this program may provide the best access to tape data.

The DT program will perform ASCII/EBCDIC translation when writing tape data and EBCDIC/ASCII translation when reading tape data if required. It will also perform standard text file reformatting, such as carriage return/line feed insertion and removal, record padding and trimming, tab expansion and contraction and record blocking and deblocking. The DT program is unable, however, to alter or vary these processing options within a single file transfer, and may be unsuitable for applications where different translation modes are required within a tape record.

In the majority of applications, the DT program provides sufficient processing capabilities to handle most transfer requirements.

4.3: TDS Software

The TDS software (Tape Device Support) is a complex set of programs that provide access to tape data by emulating a disk-type device and a character-type device within the PC-DOS or MS-DOS operating system. Thus, tape data appears as either a file on disk, accessed in the fashion that disk files are normally accessed, or as a character device, functioning in a fashion very similiar to standard DOS devices such as CON: or PRT:.

As with the DT program, translation (ASCII/EBCDIC and EBCDIC/ASCII), carriage return/line feed processing, tab expansion and contraction and record blocking and deblocking processing options are available.

4.3: (...continued)

The TDS software is controlled via the DOS IOCTL facility, which is not supported by all applications and/or all applications languages. If your application does not require interactive control of the TDS software by your program, the IOCTL program provided with the TDS software may provide all the control you need to select options and parameters within the TDS software. If more complex processing is required, check the technical manuals for your application and/or language program. Furthermore, the assumptions made by the TDS software to perform device emulation are quite complex and may not be valid for every application, especially where translation or reformatting options vary within a file. In addition, data transfers via the TDS software are limited by the maximum file size allowed by the operating system when using the disk-type device(s). With current releases of DOS 2.x and 3.x, this limit is 32 megabytes (33.5 million bytes).

Because the TDS software provides for accessing tape data in the same fashion that disk data is accessed, in most applications it provides the best access to tape data for standard application programs, high level languages and utilities.

4.4: BPS Software

The BPS (BASIC Programming Support) software may only be used with interpreter versions of Microsoft BASIC, such as BASICA and GWBASIC.

It is possible that a particular version of these BASICs may not work. If the version you have is not standard IBM BASIC 2.0, or 3.0, compatibility cannot be assured.

This software provides three entry points for the USR function that are used to transfer string data and commands to and from the tape drive.

The BPS software provides a fast and efficient programming environment for the BASIC programmer who is developing tape-specific applications. The BPS software is recommended for applications programmed in the interpretive BASIC language. In many applications, the TDS software will also work with BASIC.

4.5: The GPTR Software

The **GPTR** (General Purpose Tape Routines) routines are a set of subroutines callable from most languages that support subroutine linkage via the PC-DOS and MS-DOS LINK program. The GPTR routines are supplied in an assembly source form and as a linkable object file.

The GPTR routines assume that word pointers in the data segment are passed on the stack by the calling program. Record blocking and deblocking and translation are provided by these routines. You must be familiar with the LINK program, linkage conventions in general and the specific linkage conventions used by the assembler and your application language to use these routines. In addition, the workspace allocation method used by these routines is rather unique and requires careful consideration.

The GPTR routines are fast, efficient, effective and pose few constraints on tape data transfers. *These routines are recommended in all applications where the programmer is sufficiently skilled to utilize them and where the required linkage conventions are available.* The GPTR routines are specifically designed to be called from user-created programs, written in languages such as Lattice C, Microsoft C, Turbo-C, or Microsoft Macro Assembler. Driver programs may be written to provide linkage to these routines from software not conforming to the same linkage standards.

Chapter: FIVE

The DT Program

5.1 Introduction

The DT program is used to copy, and optionally reformat, data from the tape drive to a DOS device or file and/or from a DOS device or file to the tape drive.

The syntax of the DT program follows along the lines of the DOS COPY program.

5.2: Command Syntax

For transfers that copy disk (or other DOS device) data to tape, the command syntax is as follows:

DT [*d:*][*path*]*filename*[*.ext*] **TAPE**[*n*]:[[*parameter*]...]

For transfers copying tape data to disk (or other DOS device):

DT **TAPE**[*n*]:[[*parameter*]...] [*d:*][*path*]*filename*[*.ext*]

d:, *path*, *filename* and *.ext* conform to DOS standard nomenclature.

The parameter(s) that may be specified select various processing options, such as logical record length, translation, etc.

Each parameter begins with a "/" character. As many parameters as required may be specified. When more than one parameter is to be specified, there must be no spaces or other characters between the parameters. Additionally, there may not be any characters separating the colon following the TAPE[*n*]: specification and subsequent parameters.

One or more spaces (or other valid DOS delimiter) are used to separate the command file name DT from the file specification and from the TAPE[*n*]: specification.

The *n* optionally appearing in the TAPE[*n*]: specification is used to select which tape drive is to be used in the transfer if a tape drive other than drive 0 is to be used. To use tape drive 0 (the normal case), simply specify TAPE: or TAPE0:.

Thirteen different parameters are defined for the DT program, as follows:

/B:*n* is used to specify the physical block size used on disk to tape transfers. The block size is assumed to be equal to the record length if this parameter is not specified. If this parameter is specified, the block size *n* must be a multiple of the logical record length.

/R:*n* is used to specify the logical record length. On disk to tape transfers where /CR is specified, records read from disk are filled with trailing spaces until the record length is reached, or truncated to the record length if longer than the specified record length. The physical block size must be an even multiple of the logical record length. A logical record length of 80 bytes is assumed if this parameter is not specified.

/SF:*n* specifies the number of files to be skipped on tape before beginning a transfer.

/EOT specifies that the tape is to be positioned to the logical end of tape prior to writing the file to tape. Errors may occur when this option is specified on a new or scratch tape or when this option is specified when the tape is already positioned at end of tape.

/BOT specifies that the tape is to be rewound to the beginning of tape prior to reading or skipping to a file to be read.

/XOFF — specifies that no translation of tape data is to be performed. If /XOFF is not specified, ASCII to EBCDIC translation will be performed on transferred data prior to writing to tape, and EBCDIC to ASCII translation will be performed on transferred data prior to writing to disk.

/CR — specifies that records are delimited by a carriage return character (hex 0D) on disk. When this parameter is specified on tape to disk transfers, a record of the specified logical record length is read, all trailing blanks are removed, and a carriage return character is appended before the record is written to disk. On disk to tape transfers, records are read and assembled up to the terminating carriage return character, and padded with trailing blanks or truncated to the specified logical record length as necessary before being written to tape. Use of the /CR parameter also instructs this program to use ASCII file rules for end of file processing. On disk to tape processing, an end-of-file condition is indicated by the presence of a Control-Z (hex 1A) character in the file. On tape to disk processing, the file is filled out with Control-Z characters to the nearest 128 byte boundary.

/LF — specifies that, in addition to a carriage return character as described above, a line feed character follows a carriage return character as a record delimiter on disk files. When this parameter is specified on disk to tape transfers, line feed characters (hex 0A) are ignored when they immediately follow carriage return characters and are not to be written to tape. When used with tape to disk transfers, this parameter causes a line feed character to be appended to each record after the carriage return character before the record is written to disk. The /CR option must be used when this option is selected.

/T — specifies that tab processing is to take place. Tab characters found in a disk file copied to tape are expanded to one or more spaces such that the character following the tab character will immediately follow a space character where the space character will be in a column number evenly divisible by eight. On tape to disk transfers, a tab character will replace a space found on tape and up to seven immediately preceeding spaces where the space character occurs in a column number evenly divisible by eight. The option may only be specified when the /CR option is also selected.

/A — specifies on disk to tape transfers that an existing tape file is to be extended by backspacing the tape over the preceding file mark. This repositioning is performed after all other positioning is completed, such as rewinding (/EOT) or file skipping. Use of this feature may result in a file containing a short block followed by a block of normal length. This situation is handled correctly by most operating systems. On tape to disk transfers, this option specifies that the disk file is to be appended, as opposed to being erased and rewritten.

5.2: (...continued)

/SHORT specifies that short records are permitted. This option is used to avoid the restriction that the length of physical blocks found on tape to disk transfers must be even multiples of the record length. This option has no meaning for disk to tape transfers.

/U specifies that the transfer is to take place independent of logical record length. All data is transferred on a byte for byte basis. Carriage Return and/or Line Feed processing may not be used with this option. This option is recommended on tape to disk transfers where variable length tape records are used, or to transfer data where correct record blocking on tape need not be verified. It may be used on disk to tape transfers to create a binary image of a file, such as for for backup or disk file transfers. When this option is specified, the record length parameter must not be specified and the block size is assumed to be 2K (2048) on disk to tape transfers unless otherwise specified.

/RECORDS:*n* may be used to specify the maximum number of records that are to be transferred. If this parameter is not specified, all records in a disk file will be transferred to tape, while tape to disk transfers will proceed until a file mark is encountered.

/X:*filespec* allows the use of externally defined translation tables. These translation tables must be 512 bytes long, with the first 256 bytes containing the corresponding ASCII values to replace EBCDIC characters, and the next 256 bytes containing EBCDIC values to replace ASCII characters. The ASCII.ASM and EBCDIC.ASM source files used to create sample translation tables are included in this release. The external translation tables are specified by using a /X:*filespec* in the command line of the utility or installation program. The filespec contains a file name and optionally a *disk* drive designator, *path*, and/or *extent*.

5.3: General Operations

Tape positioning is performed in three phases. First, if /BOT is specified, the tape is rewound to the beginning of tape mark. Second, if /EOT is specified, the tape is then spaced forward until two consecutive file marks are encountered, or, if /SF:*n* is specified, the tape is spaced forward until the specified number of file marks *n* have been skipped over. If two consecutive file marks are encountered while skipping where the /SF:*n* has been specified, an error is returned and no data is transferred.

The third step of positioning occurs if the /A (Append) parameter is specified. This causes the tape to be backspaced one file mark, to allow the preceeding file to be extended, or appended. Note that when this method of extending a file is used where tape records are blocked, (i.e., the blocksize is a multiple of the record length), the preceeding file may end in a short block, which may then be followed by a block of normal length. This condition is considered acceptable by most operating systems, however, some operating systems and tape management programs may prove incompatible with this method.

It is recommended that the /BOT parameter be specified in most situations, as prior usage of the tape may have left it in an unknown or undesirable position. Aborted attempts at running the DT program, for example, may leave the tape one or more blocks into a file. Be sure that the current position of the tape is known and correct when not using the /BOT or /EOT parameters.

Transfers are performed on a logical record basis. On disk to tape transfers, a record of disk data (up to the next carriage return character, if /CR is specified, or a record of length equal to the logical record length if /CR is not specified) is read, then moved to the tape buffer. If tab processing is specified (with the /T parameter), tab characters in the text are replaced with the appropriate number of spaces. Padding blanks are added to fill out the record if it is too short, and excess characters are truncated if it is too long. This process is repeated for each record until the tape buffer is filled up to the specified physical block size, which causes the tape buffer to be written to tape.

On tape to disk transfers, a record of tape data equal to the logical record length is removed from the tape buffer. If the /CR option is specified, trailing blanks are removed and a carriage return (and optionally a line feed) character is appended to the record. Spaces occuring on tab boundaries and the preceeding spaces are replaced by tab characters if the tab option (/T) is specified. The resulting record is then written to disk. This is repeated for each record in the tape file.

When unformatted records are transferred with the /U parameter, or when the /SHORT option is specified, the logical record length is temporarily set to be compatible with whatever physical block size is found on tape.

Values for the logical record length (specified by the *n* in the /R:*n* parameter), the physical block size (the /B:*n* parameter) and the file skip count (the /S:*n* parameter) must be specified in the command line as decimal values, without commas, in the range of 0-65535.

Not all possible combinations of parameters are allowed. The /CR and /R:*n* options may not be selected when the /U option is selected. The /B:*n* option may not be selected for tape to disk transfers, as the physical block size used by this software when reading from tape is the block size that is actually found on tape.

After a disk to tape transfer has been completed by this program, two consecutive file marks are written on the tape, and the tape is then backspaced over one of these file marks. This results in the tape being positioned so that another file may be added to the tape by simply executing this program again, without specifying any positioning parameters, or the file which has just been written to tape may be appended by specifying only the /A positioning parameter on a subsequent execution.

The positioning options must be carefully selected by the user for validity in each application or error conditions may result. One example of this would be specifying /EOT on a new tape, where no logical end-of-tape mark has been written. Another example would be specifying the /A parameter when the tape has just been loaded and is already at beginning of tape, leaving no preceeding file mark to be spaced over and no preceeding file to be appended.

If a physical end-of-tape is encountered by this program while writing data to tape, two file marks are written and the tape is rewound and unloaded. The operator is then prompted to mount a continuation volume, and the disk to tape transfer continues.

5.4: Examples Using DT

To transfer 80-byte (default logical record length) records from tape to disk, translating from EBCDIC to ASCII, without reformatting, to a disk file named DATA.FIL, the following command would be used:

```
DT  TAPE:  DATA.FIL
```

To transfer the same data, except with tab compression, trailing blank removal and carriage return and line feed insertion:

```
DT  TAPE:/CR/LF/T  DATA.FIL
```

To transfer a binary image of a disk file (BIN.FIL) to tape, using 1024-byte records, the following would be used:

```
DT  BIN.FIL  TAPE:/XOFF/B:1024/U
```

To append the last (or only) file on tape with 133-byte fixed blocked records, ten records to the block, converting carriage return/line feed terminated ASCII records from the file THIS.LST to fixed length right padded EBCDIC records:

```
DT  THIS.LST  TAPE:/EOT/A/CR/LF/R:133/B:1330
```

To transfer the first one hundred lines of a source file to tape using blocks of ten eighty-byte records, writing the records in EBCDIC:

```
DT  TRANSACT.C  TAPE:/B:800/RECORDS:100/CR/LF/T
```

Chapter: SIX

The TBACKUP, TRESTORE and TDIR Programs

The **TBACKUP** and **TRESTORE** programs are used to create and retrieve backup images of disk files. Directories of these tapes may then be retrieved via the **TDIR** utility.

6.1 General Information

The syntax of the TBACKUP and TRESTORE programs closely follow that of the DOS BACKUP and RESTORE commands. Some enhancements have been provided to take advantage of the increased capabilities of the 1/2" magnetic tape.

User classification of files is provided to allow the user to selectively restore groups of files from a tape when multiple backups are performed to a single tape. As an example, the user could assign a different class code for each day of the week and simply append his prior day's backup tape with today's data. If a version of a file (or files) from earlier in the week is needed, the class code could then be used to select the desired day's version.

6.1: (...continued)

The standard BACKUP operation has also been extended to allow for backing up more than one disk in a single operation. This is particularly useful in situations where a single physical disk drive has been partitioned into more than one logical disk.

The DOS BACKUP's archive bit support and modification date support have been preserved in the TBACKUP program. During a normal backup operation, the archive bit of each file that is successfully backed up is reset. However, this prevents successful operation on write-protected disks or diskettes. Therefore, an option has been added to bypass setting the archive bit. This option is also useful in preserving the archive bit if multiple backup methods are being used on a disk.

Additional extensions have been made to the capabilities of the TBACKUP program to allow the operator to specify a file or group of files that are to be excluded from a backup operation. This allows a disk to be backed up which contains a file or files that are unreadable, providing such files can be isolated in a single global filename specification. (This is usually accomplished by renaming the files to be excluded so that they all have a common extension, such as .BAD, and then excluding all files that match the global filename *.BAD).

The operator may also specify that the TBACKUP operation is to exclude all files to which the operating system denies access, allowing files that are currently being "shared", for example, to be skipped over without causing the backup to be aborted.

6.2: The TBACKUP Program

The TBACKUP program is used to make a backup tape containing images of one or more files from one or more disk drives. It should be noted that these are images of disk files, not images of disk drives as used with some other backup devices. These images contain no disk structure information and may be retrieved as required independently of the disk drives structure and integrity. All disk accesses are made through the operating system and are not dependent upon particular directory structures or knowledge of internal control blocks. This method provides a much greater degree of compatibility and utility than "mirror image" backups.

6.2.1: TBACKUP Command Syntax

The command syntax for the TBACKUP program is as follows:

TBACKUP [*d*:][*path*][*filename*[.*ext*]] **TAPE**[*n*]:[[*parameter*]...]

d:, *path*, *filename* and *.ext* conform to DOS standard nomenclature.

If no drive (*d:*) is specified, the default drive is used. Likewise, if no *path* is specified, the current directory is assumed. If no *filename* is specified, all files in the entire current (or specified) directory (and subdirectories, if specified) are backed up, as if *.* were specified for the *filename* and *extension*. Global filename characters ('?' and '*') are allowed in the *filename* and *.ext* and cause all matching files to be backed up.

Each parameter begins with a "/" character. As many parameters as are required may be specified. When more than one parameter is to be specified, *there must be no spaces or other characters between the parameters. Additionally, there may not be any characters separating the colon following the TAPE[n]: specification and subsequent parameters.*

One or more spaces (or other valid DOS delimiter) are used to separate the command name TBACKUP from the file specification and from the TAPE[*n*]: specifier and its parameters.

The *n* optionally appearing in the TAPE[*n*]: specification is used to select which tape drive is to be used in the backup if a tape drive other than drive 0 is to be used. To use tape drive 0 (the normal case), simply specify TAPE: or TAPE0:

Twelve different parameters are defined for the TBACKUP program, as follows:

/S specifies that all subdirectories of all levels below the specified (or current, if no directory is specified) directory are to be backed up as well as the current directory.

/M specifies that only files marked as having been changed since the last time they were backed up are included in the backup operation. The TBACKUP program resets the archive bit in each file's directory as it is backed up. The DOS BACKUP program also resets this bit, and care is necessary to make sure that files are backed up properly when using this parameter with either the TBACKUP program or the DOS BACKUP program. (The archive bit in a file's directory entry is automatically set by the operating system whenever data is written to the file and the file is closed).

/A specifies that the tape is to be positioned at the logical end of tape prior to commencing the backup operation, causing the tape to be appended. If this parameter is not specified, the tape is rewound to the beginning of tape prior to the backup operation.

/D:*mm-dd-yy* is used to specify that only files that have been written to on or after the specified date are to be backed up.

/FAST specifies that the high speed of the tape drive is to be selected on all tape commands. In STREAMing mode, (/STREAM parameter selected) this generally decreases the time required to perform a backup operation by as much as fifty percent, but correspondingly increases the amount of tape used by the backup operation to more than double that used when this parameter is not selected. When STREAMing mode is not specified, this command may or may not decrease the backup time, depending upon the particular data file sizes and organization being backed up, as the tape repositioning time is quadrupled as the speed is quadrupled. Use of this parameter is not recommended for normal operation.

/STREAM when used with tape drives that allow extension of inter-record gaps, specifies that tape motion is to continue and extended inter-record gaps are to be written during the backup operation while disk accesses are taking place. Use of this parameter decreases the amount of time required to perform the backup by eliminating the normal repositioning time taken when data is not ready for the tape drive after the tape drive has completed the prior operation. Use of this method increases the length of tape used on a given backup operation by approximately 30 percent at low speed and as much as 120 percent at high speed. In addition to causing the extension of inter-record gaps, this parameter also causes tape and disk accesses to be alternated, as with the /ALT parameter described below. Use of this parameter is not recommended in normal operation.

/C:*n* assigns a class code to all files backed up to the tape by this backup command. The class code is a number in the range of 1 to 65535. This code may be used, for example, with multiple TBACKUP commands and the /A option to distinguish which version of a file is to be selected for restoration when multiple versions of the file are backed up to the same tape.

/DISKS:*string* specifies a list of disk drive(s) to be backed up. This option is used to back up more than one disk drive with a single command. The string is a list of disk drive designators of the disk drives to be backed up. The *d:* drive specification must not appear in the command line with the *path* and *filename* when this option is used. As the path specification is still in effect, care must be taken to specify a path that exists on each disk drive to be backed up. If no path is specified, the current directory (and its subdirectories, if so specified) of each drive is backed up.

/RO specifies that the archive bit in the directory of files that are backed up is not to be updated. *This allows disks that are write-protected to be backed up.*

/I specifies that the TBACKUP program is to ignore files to which access is denied by the operating system. When access is denied to a file and this option is specified, a message is printed on the standard error device (usually CON:) indicating the file that was bypassed and the backup operation continues.

/EX:[*d:*][*path*][*filename*][*.ext*] is used to specify files that are not to be backed up by the TBACKUP program. The file specification may include the global filename characters "*" and "?". If no *filename* is specified, this option functions as if "*.*" were specified as the *filename* and *extension* to be excluded. Likewise, if no *path* or *disk* is specified, matching files in all directories and disks are not backed up. To exclude multiple files, specify the filenames with a semicolon ";" separating each.

/ALT specifies that the TBACKUP program is to alternate tape and disk operations. This may be necessary when backing up disk drives that require continuous DMA operations, or when the controller for the tape drive used to back up the disk shares the same DMA channel as the disk drive being backed up. In normal operation, the TBACKUP operation performs disk read operations concurrent with tape write operations to reduce the total time required to back up a disk. Use of the /ALT parameter will slightly increase the time taken by the TBACKUP program to back up a given amount of data.

6.2.2: TBACKUP General Operation

The TBACKUP program attempts to back up all disk files in the specified directory, or current directory if none is specified, that match the specified *filename* (unless excluded via the /EX parameter). If the /S parameter is specified, all matching files in all subdirectories of the specified or current directory are also backed up.

The /M and /D:*mm-dd-yy* options are used to select files for backup that have either been modified since the last backup operation (the /M parameter), or files that have been modified since a specified date. The date and archive bit information in the directory, provided by the operating system, is used to select these files. Dates may be specified in accordance with the rules for the DOS BACKUP and DATE commands. The TBACKUP program accepts dates in the form *dd-mm-yy* on systems configured for European dates, as indicated by the operating system's "Return Country Dependent Information" call.

As with the DOS BACKUP command, an existing backup can be appended. As described in the TINSTALL program, one of two options can be selected by the TINSTALL program to control whether or not tapes are automatically appended. The standard method used by the TBACKUP operates in a manner very similiar to the DOS BACKUP, with backup tapes not being appended unless the /A option is specified. The alternate method used by the TBACKUP program assumes that TBACKUP tapes are always to be appended unless they have been scratched via the SCRATCH program. With either of these methods, a tape must contain a valid TBACKUP beginning and ending label to be appended. (The beginning label is not checked if an append is to take place, the tape drive has been continuously online since the last operation and the tape contains a valid TBACKUP ending label). If the mounted tape is neither a valid TBACKUP tape nor a SCRATCHed tape and an automatic append is to take place, the operator is prompted before the tape is used for backup.

When using the standard method, if the /A option is not specified the tape is rewound and a tape exercise is performed verifying correct tape drive operation by writing, backspacing and reading a standard data pattern to and from the tape drive prior to commencing with the backup. Upon successful completion of this test the tape is rewound, a label is written and the backup operation is initiated. If the /A option is specified, it is verified that the tape has not been taken offline since the last tape operation and the tape is checked to make sure that it is positioned at the logical end of tape. If the tape is not at end of tape or if the tape has been taken offline since the last operation, the operator is prompted for a response before the tape is rewound and positioned at end of tape.

The tape image of the disk file consists of a special label record followed by the actual disk data, if any. (To ease the restoration of an entire disk, empty disk files are also backed up to tape, as are hidden files and system files).

If an end-of-tape condition is encountered by this program, a file mark followed by an end-of-volume label and two more file marks will be written to tape to inform a subsequent TRESTORE program that the backup operation requires an additional reel. The tape is then unloaded and the operator prompted, requesting an additional reel of tape. When the new reel has been mounted and the tape drive is online, the operator should respond to the prompt by pressing the letter **Y** followed by a carriage return and the backup operation will continue where it left off.

As matching files are found by the TBACKUP program, the *disk*, *path*, *filename* and *extension* are listed on the standard output device, (normally CON:). This listing may be redirected using the DOS piping facility, or suppressed by redirecting it to the NUL device. Redirecting the listing to the NUL device will decrease the amount of time needed to backup a disk, but gives the operator no way to tell if the tape drive has runaway.

If access to a file is denied, the *disk*, *path*, *filename* and extension are listed on the error device (also normally CON:) together with a message indicating that access was denied.

6.3: The TRESTORE Program

The TRESTORE program is used to retrieve images of disk files from backup tapes created with the TBACKUP program, write the data from the file image to disk, and update the file's date and attribute information to match that which existed when the disk was backed up.

6.3.1: TRESTORE Command Syntax

The command syntax for the TRESTORE program is as follows:

TRESTORE TAPE[*n*]:[[*parameter*]...] [*d*:][*path*][*filename*[.*ext*]][/P][/S]

d:, *path*, *filename* and .*ext* conform to DOS standard nomenclature.

If no drive (*d*:) is specified, the default drive is used. Likewise, the current directory is assumed if no *path* is specified. If no *filename* is specified, all files are restored as if *.* were specified for the *filename* and *extension*. Global filename characters ('?' and '*') are allowed in the *filename* and .*ext* and cause all matching files to be restored. If a *filename* containing global filename characters (or no file name at all) is specified, the restore operation searches the entire tape for all matching files until a logical end of tape is encountered. If a specific (non-global) *filename* is specified and files in subdirectories are not to be restored, the restore operation ceases immediately after restoring a single file.

Each parameter begins with a "/" character. As many parameters as are required may be specified. When more than one parameter is to be specified, there must be no spaces or other characters between the parameters. Additionally, there may not be any characters separating the colon following the TAPE[*n*]: specification and subsequent parameters.

One or more spaces (or other valid DOS delimiter) are used to separate the command name TRESTORE from the TAPE[*n*]: specification and its parameters and, in turn, to separate these from the file specification.

The *n* optionally appearing in the TAPE[*n*]: specification is used to select which tape drive is to be used in the restore if a tape drive other than drive 0 is to be used. To use tape drive 0 (the normal case), simply specify TAPE: or TAPE0:

Five parameters are defined that may follow the TAPE[*n*]: specification in the TRESTORE program:

/FAST specifies that the high speed of the tape drive is to be selected when the tape is being searched for matching files. Specifying this parameter results in a seventy-five percent reduction in search time, however, it also results in a substantial increase in repositioning time each time a matching file is found, which in certain circumstances increases the duration of the restore operation considerably. Use of this parameter is recommended in most circumstances if your configuration allows high speed transfers. The high speed search can also be selected automatically by setting the high speed search bit in the TCONFIG.SYS file. (*This is described by the TINSTALL program*).

/C:*n* is used to select a class code to limit the files being restored to those with a matching class code. This is a code in the range of 1 to 65535 written in the header information by the TBACKUP program. If this parameter is not specified, the class code is ignored by the restore operation.

/DISK:*d* is used to specify a disk drive designator to be used for the purpose of selecting which of the files on the tape are to be restored. This parameter may be used to restore files to a disk other than the disk they were backed up from originally. The disk drive designator *d* specifies the original disk drive from which the backup tape was made which is to be searched for matching paths and *filenames*. Matching files are then restored to either the current default disk drive, if none is specified with the *filename* specification, or the the disk drive specified with the *filename*. If this parameter is not specified, files may only be restored to the disk drive from which they were originally backed up.

/PATH:*path* is used to allow files to be restored to a directory other than the one they were backed up from originally. This is best thought of as restoring files from one path to another. Files on tape originating from the path specified in the /PATH parameter are restored to the path specified with the *filename* specification, if one is given, or to the current directory if no disk path is specified. Thus, the path specified in the /PATH parameter is used only for the purpose of determining whether or not a file on tape is to be restored. This path is then replaced by the path to the current disk (or *path* specified with the *filename* in the command line) when the file is restored to disk. The path specified in the /PATH parameter replaces the path from the root directory to the node specified by the *filename* in the command line (or implied by the current directory, if no *filename* is specified). Branches of the path beyond the specified nodes are assumed to be identical between the path on tape and the path of the disk to which files are being restored. The path used with this parameter must begin with the root directory (and start with a backslash "\") and not contain any relative references (such as "..").

/DISKS:*string* is used to specify a list of disk drives to be restored. This parameter cannot be used in conjunction with the /DISK:*d* parameter or when a disk drive is specified in the command line. The /DISKS parameter is used to allow the files from more than one disk drive to be restored at a time. The files will be restored to the same disk they were originally on when the backup tape was made.

In addition, either or both of the two following parameters may follow the disk file specification in the command line:

/S specifies that all files found on the tape that were backed up from sub-directories of the specified (or current, if no directory is specified) directory that match the specified *filename* are to be restored. If this parameter is used, the restore operation searches for all matching files until a logical end of tape is encountered.

/P is used to specify that a prompt is to be issued and a reply required each time the restore operation encounters a file on disk that is to be replaced which has an attribute byte identifying the file as R/O (Read Only) or as having been modified (as indicated by the Archive Bit) since the last BACKUP or TBACKUP operation took place. The prompt requires that a reply of **Y** or **N** followed by a carriage return be entered from the console by the operator, with the **Y** response indicating that the file is to be restored and the **N** response indicating that the file is not to be restored.

6.3.2: TRESTORE General Operation

When the TRESTORE program is executed, the tape is rewound and the label records are read, displaying selected information upon the system console. The volume sequence number is checked to verify that the tape mounted is the first reel of the backup, in case a multi-reel backup was made. An opportunity is given to override the sequence on a restore operation if the first tape reel, or subsequent reels, are not mounted in order.

The TRESTORE program scans through the backup tape, attempting to match the label information of each file on tape to the criteria specified in the command line. The *disk* drive, *path*, *filename*, extention and class (as set by the TBACKUP command) all play a part in this search. If a *filename* is not specified on the command line, all *filenames* found on tape will be considered matching. The same applies to the class. TRESTORE will only match files backed up from the currently defaulted disk drive if no disk is specified or overridden via the /DISK or /DISKS parameter. It will match files backed up from the current directory if no path is specified in the command line or overridden via the /PATH parameter.

The DOS system files IBMBIO.COM and IBMDOS.COM are hidden files in the root directories of boot disks that contain the operating system and BIOS information necessary for the system to boot. These files must appear in reserved areas, and are written there by the DOS SYS command.

Although these files are backed up to tape (as are all system and hidden files), if they are restored from tape they are not only not written to the reserved areas, but replace the information in the directory that must point to the image in reserved areas, leaving the disk unbootable. If you are restoring the root directory of your boot disk, you must be careful to not restore these files!

You can normally avoid restoring these files by specifying the /P option (requesting a prompt before rewriting files that are set to read-only mode), and replying N to not restore these files when they are encountered on tape. You may also wish not to back these files up to begin with, by specifying /EX:\IBM*.COM as a parameter in the TBACKUP command to exclude these files.

If subdirectory restoration is not enabled (with the /S parameter) and a *filename* containing no global characters ("*" and/or "?") is specified, the restore operation will stop after restoring a single file. Otherwise, the search proceeds through the entire tape, restoring all matching files.

Files are restored to the same status, creation/modification date and attributes that existed when they were backed up. Empty files are restored as zero length files.

A path must exist to the file or path specified in the command line for the disk file(s), however, it need not exist for all subdirectories below that node containing files to be restored. The TRESTORE program will create subdirectories as needed once the original directory or subdirectory is located. This feature facilitates restoration of an entire disk, as the disk only need be formatted and then restored with a single command.

The /PATH and /DISK parameters facilitate transferring data from one computer to another, particularly in situations where the disk configurations are not identical. The disk drive and path of the originating disk can be replaced with another disk drive and path to any node for the purpose of restoration using these two parameters. These parameters are also useful when reorganizing a disk's structure during restoration.

As matching files are found by the TRESTORE program, the disk, path, *filename* and *extension* are listed on the standard output device, (normally CON:). This listing may be redirected using the DOS piping facility, or suppressed by redirecting it to the NUL device. Redirecting the listing to the NUL device will decrease the amount of time needed to restore from a tape, but gives the operator no way to tell if the tape drive has runaway.

If the tape is unreadable or a file is marked as a read only file or has been modified since the last backup and the /P option has been specified, the *disk*, *path*, *filename* and *extension* of the file are listed on the error device (also normally CON:) and the operator is prompted for a response.

6.4: The TDIR Program

The TDIR program is used to obtain a listing of file images appearing on a tape created by the TBACKUP program. A listing similar to the one generated by the DOS DIR command is created by this program. This listing, however, can span multiple "directories" and contains attribute information not listed by the DOS command.

6.4.1: TDIR Command Syntax

The command syntax for the TDIR program is as follows:

TDIR TAPE[*n*]:[*d*:][*path*][*filename*[*.ext*]][/S][/FAST] [*dev*:]

One or more spaces (or other valid DOS delimiter) are used to separate the command name TDIR from the TAPE[*n*]: specification and its file specification and parameters and, in turn, to separate these from the *dev*: specification.

The *n* optionally appearing in the TAPE[*n*]: specification is used to select which tape drive is to be used by the restore if a tape drive other than drive 0 is to be used. To use tape drive 0 (the normal case), simply specify TAPE: or TAPE0:

The *d*:, *path*, *filename* and *.ext* following the TAPE[*n*]: specification must immediately follow the TAPE[*n*]: specification and may be used to limit the directory search to a particular *disk* drive, *path* and/or *filename* and optional *extension*. Global filename characters ("*" and "?") may appear in the *filename* and *.ext* and cause all files with matching *filenames* and *extensions* to be displayed by the directory search. If a *path* is specified it must begin with a backslash indicating it represents a path from the root directory and must not contain any relative path specifications, such as "..".

/S indicates that all subdirectories of a specified path are to be listed, as well as the directory indicated by the path itself.

/FAST causes the tape search to be performed at the tape drive's high speed and defers the listing of the tape directory until the search has been completed. This usually results in a very significant reduction in search time. This high speed search may also be invoked automatically when the high speed search bit is selected in the TCONFIG.SYS file. (This option is described by the TINSTALL program).

If both the /S and the /FAST parameters are specified, they may appear in either order.

The *dev*: specification is used to specify the device or file to which the directory listing is to be sent. If no value is specified, the standard output device (usually CON:) is used. The *dev*: specification may consist of a character device name, such as LPT1, optionally followed by a colon, or may be disk file specification, conforming to DOS standard nomenclature, consisting of an optional *disk* drive, *path*, *filename* and/or *extension*.

6.4.2: TDIR General Operation

Each separate disk directory in which matching files were found by the TDIR search is displayed separately, with the matching files within the directory listed after the *path* to the directory and a header. A filename, size, date, time, four attribute flags, and a TBACKUP class are listed for each matching file found in the directory.

Letters corresponding to the file attribute bits, R for Read Only, H for Hidden File, S for System File and A for Archived File are displayed at the end of each file entry.

If the /FAST parameter is not specified in the command line and the high speed search bit is not selected in the TCONFIG.SYS file, matching file entries are listed as they are located on tape. If this is not the case and a high speed search takes place, the TDIR program reads up to 64K bytes of directory records (about 700 file entries) before scanning back through and listing matching entries. If the directory operation is not complete, another batch is then read and the operation continues.

If the *dev:* specification does not appear on the command line, the output of the TDIR program is sent to the standard output device, which may be redirected via the DOS piping facility.

6.5: Examples Using TBACKUP, TRESTORE and TDIR

To back up the entire contents of the current disk to tape, use the following:

TBACKUP *.* TAPE:/S

To restore the entire contents of a disk drive from a backup tape, such as that created with the preceeding command, use:

TRESTORE TAPE: *.*/S

To create a backup tape of all files in the current directory (only):

TBACKUP *.* TAPE:

To restore those files to the current directory that have a .COM extention:

```
TRESTORE    TAPE:    *.COM
```

To back up disk drives A, B, and E to tape:

```
TBACKUP  \*.*  TAPE:/S/DISKS:ABE
```

To take all of the files that were backed up from disk drive and path E:\DT\DIST and transfer them to disk drive and path C:\MASTER\LIB\DIST:

```
TRESTORE    TAPE:/PATH:\DT\DIST/DISK:E     C:\MASTER\LIB\DIST\*.*
```

To back up all files that have been modified since the last backup, designating them as class 5:

```
TBACKUP  \*.*  TAPE:/C:5/S/M
```

To list a directory of all files on a backup tape to the console:

```
TDIR    TAPE:
```

To list to the printer a directory of all files having an extension of .COM that were backed up from disk drive B:

```
TDIR   TAPE:   B:*.COM LPT1:
```

Notes

Chapter: SEVEN

The TDS Software

7.1 Introduction

The TDS (Tape Device Support) software provides an operating system environment with access to 1/2" tape data through operating system (DOS or MSDOS) devices. The TDS software, in addition to providing simple data transfer capabilities, also performs data manipulation and reformatting for the purpose of transcribing data between the differing contexts normally associated with 1/2" tape data representations and microcomputer operating systems' data representations.

The TDS software provides most system software, utilities, and languages capable of performing I/O through the operating system with access to 1/2" tape data in conventional, fixed blocked data interchange formats without modification.

7.1: (...continued)

The TDS software consists of a primitive device driver, the main device driver and the IOCTL utility program.

The TDS device drivers have been split into two sections, a primitive device driver and a main device driver. This is done to provide flexibility to the user in what would otherwise be a cumbersome system. Because the TDS software can occupy as much a several hundred thousand bytes of memory or as little as 15K, depending upon the number of tape drives installed and the maximum blocksize permitted, it is convenient to be able to dismiss and reinstall the main device driver at will, freeing up memory for other uses such as word or spread sheet processing when the TDS software is not in use. This alleviates the need to have two boot disks available: one for applications requiring large amounts of free memory, the other for tape transfers.

7.2: Installing the TDS Primitive Device Driver

In order for the TDS software to function, the TDS.DEV primitive device driver must be installed as indicated in section 2.4 of this manual.

The TDS.DEV file is broken up into three parts. The first part is the resident portion. This contains a very small program that is merely able to tell the operating system that it exists and link itself to the large, sophisticated device driver TDS.COM actually used to provide data transfer. The second portion of the TDS.DEV device driver is the initialization portion, which initializes the controller(s) and fills out the system control blocks used to implement the logical devices from the sizing parameters. The final portion is the image of the TCONFIG.SYS file from which the sizing parameters and controller addresses are extracted. The TDS.DEV primitive device driver will interpret a parameter on the **DEVICE=** command line in the CONFIG.SYS file to configure the software for fewer than eight block and character devices. Placing a /D:*n* specification at the end of the **DEVICE=** command will limit the driver installation to the drive matching the value for *n*, which must be 0-7. This syntax is only allowed by DOS 3 and up.

Although the TDS.DEV file is relatively large (4K+ bytes) only a small portion of it (roughly 1/2K) actually remains resident and occupies memory.

7.3: Installing the TDS Main Device Driver

In order to use the TDS devices you must install the main device driver before you actually require the use of them. Unless you use these devices with all of the applications you run on your computer, you will probably want to install the TDS main device driver each time you need it immediately prior to use, and dismiss it (via the FREE utility) as soon as you are done with the tape transfers. If, however, you are using the TDS software almost exclusively on the system, or wish to have it fully and automatically installed upon each bootup of the system, it is possible to place a TDS command in the AUTOEXEC.BAT file on the root directory of your boot disk.

The main module of the TDS software is provided as the file TDS.COM on the distribution diskette. When this module is executed as a program, it installs itself and returns control to the operating system. (The DOS "Terminate and Remain Resident" system call is used for this installation).

The command syntax for installing the TDS software is as follows:

TDS [/D:*n*][/M:*n*][/S:*n*][/X:*filespec*]

Three optional parameters, /D, /M, and /S may appear in any order and are used to select the number of tape drives to be installed for use by the TDS software, the maximum blocksize for which buffers must be allocated and the number of sectors that are to be buffered in memory.

/D:*n* specifies the number of tape drives to be installed by the TDS program. The value *n* contains the logical tape drive number of the "highest" tape drive installed on the system that is to be used by the TDS software, and must be in the range of 0-7, indicating one to eight tape drives, respectively. If the /D parameter is not specified, one tape drive is installed, corresponding to logical TAPE0:.

/M:*n* selects the maximum blocksize for TDS transfers. The value *n* specified with this parameter must be a decimal number, without commas, in the range of 1-65535. If this parameter is not specified, a maximum blocksize of 32,768 bytes (32K) is assumed. The physical blocksize occuring on tape used by the TDS installation must not exceed this value. One buffer of this size is allocated from main memory for each tape drive installed. The actual amount of memory allocated for buffering may exceed this amount, however, because DMA restrictions require that buffers are located between physical 64K boundaries. Increasing this value will decrease the amount of time needed for data transfer, but will use more memory, so consideration must be taken for this.

/S:*n* specifies the number of sectors that are to be buffered in memory. If a value other than one is specified, a ring buffering algorithn is used to attempt to eliminate sequence errors when non-sequential accesses are made to the *pseudo disk* data areas. Use of this parameter is recommended in situations where other software performs its own sectoring for compatibility with DOS 1 or for optimizing disk access. In this case, the value specified should be greater than or equal to the maximum number of sectors that may be so buffered.

The need to utilize the ring buffering is normally indicated by the TDS software returning error code 21 or 22 (sector sequence errors) when interrogated via the IOCTL facility after DOS has returned a disk I/O error. In most cases the user should then attempt to remove TDS via the FREE program and then reinstall TDS using a sector buffer specification of /S:8. This will eliminate the sequence errors in many cases.

/X:*filespec* allows the use of externally defined translation tables. These translation tables must be 512 bytes long, with the first 256 bytes containing the corresponding ASCII values to replace EBCDIC characters, and the next 256 bytes containing EBCDIC values to replace ASCII characters. The ASCII.ASM and EBCDIC.ASM source files used to create sample translation tables are included in this release. The external translation tables are specified by using a /X:*filespec* in the command line of the utility or installation program. The filespec contains a *filename* and optionally a *disk* drive designator, *path*, and/or *extent*.

If either the maximum blocksize or number of tape drives *installed* by the TDS program must be changed, the current installation of TDS can be removed, using the FREE program described later, and TDS reinstalled using the new values.

If the TDS program has already been installed and a new TDS command is entered, the operator is notified of this and asked if an additional installation is to be made. (Normally this is not desirable, as the prior installation will continue to occupy memory but will be inaccessable).

A large part of the installed portion of the **TDS.COM** main device driver is the image of the DOS File Allocation Table (FAT). This table will normally be several thousand bytes long, occupying 1-1/2 or 2 bytes for each cluster of the *pseudo disk*. If you increase the cluster size via the TINSTALL program, the amount of space occupied will correspondingly decrease. If you are using DOS version 2, however, you should not increase the cluster size to more than double its normal value, as there are errors in DOS 2.x which fail to close files properly when the cluster size gets too large. Doubling the cluster size can be accomplished by modifying the cluster sizing parameter in the **TCONFIG.SYS** file and the **TDS.DEV** file through the use of the TINSTALL program to a value of 1. (A value of 10 should be keyed in, as the low order nibble is used for something else and must be zero).

Because DOS allows a maximum of 65,536 sectors (64K) of 512 bytes (1/2K) each in a logical disk drive, the maximum file size for TDS disk-type files, hence the maximum amount of data that may be transferred between a single open and close, is 64K x 1/2K, or 32M (about 33.5 million) bytes. (If the maximum sector size in future versions is increased, the corresponding maximum disk file size should also increase. This is anticipated by the sector size multiplier in the **TCONFIG.SYS** configuration file). This size of 32M is automatically allocated for DOS 3 implementations of TDS, as there are no additional limits. With DOS 2, however, there is an additional limit of 4K cluster entries in the file allocation table. Therefore, to achieve the 32M capacity available, you must increase the cluster size above the standard value of 4K to 8K, as explained in the preceeding paragraph.

7.3: (...continued)

If the cluster and sector sizing parameters in the **TCONFIG.SYS** file used by the **TDS.COM** file when it is installed do not agree with the values for these parameters that were found in the **TDS.DEV** file included in the system boot, using the TDS disk-type devices may cause the system to crash until the discrepancies in the values of the parameters are reconciled.

The TDS software uses a software interrupt vector to access the **TDS.COM** main device driver from the **TDS.DEV** primitive device driver. This interrupt, 66H, appears at the address 198H through 19BH in the lowest segment of memory. This interrupt, normally not used by DOS, must be available for this purpose and neither it, nor the corresponding memory locations, may be used by other software in the system.

7.4: Examples of the TDS Command

To install TDS software support for one tape drive with a maximum block size of 32K bytes (the default values) use the following command (*assuming that the primitive device driver has been correctly installed on the boot disk*):

```
TDS
```

To install two tape drives, addressed as drives one and zero, allowing a maximum physical block size of 65,535 bytes (64K - 1):

```
TDS  /D:1/M:65535
```

To install a single tape drive with a maximum physical block size of 8K bytes:

```
TDS  /M:8192
```

7.5: General Operation

With the TDS software fully installed, it is possible to look at the directory of the *pseudo disk* representing tape data. If the highest disk drive on your system before installing the TDS software was drive C:, the *pseudo disk* representing the first tape drive on the system is drive D:. To view the directory of this drive, simply look at the directory as you would any other disk directory:

```
DIR  D:
```

The system will respond, showing a volume name of TDS V*x.x*, a single file named TAPE, with a length of roughly 32M (about 33-1/2 million) bytes (or 16M for DOS 2 unless the cluster size has been modified), and no free space. Reading the file (D:TAPE) with a tape mounted in the drive and the drive placed online will read in translated data as fetched from tape. In addition to this *pseudo disk* device, a character device, TAPE0: also represents this same tape drive and data.

For the remainder of this chapter, it will be assumed that the first TDS *pseudo disk* is device D:. In your specific application, this may not be the case and you should interpret the D: in the examples in this manual to represent whichever disk device this actually represents on your system.

In your program or utility, you are free to use either the *pseudo disk drive* or the character device to fetch and/or store data to and from the tape drive.

If you have a sample tape containing 80-byte EBCDIC data records you wish to display, you may use the DOS system utilities COPY or TYPE to view the contents of the tape by entering:

COPY TAPE0: CON:

or

TYPE TAPE0

or from the *pseudo disk*,

COPY D:TAPE/A CON:

or

TYPE D:TAPE

(If any of the above commands do not appear to function correctly, it may be due to tape data unsuitable for display or to other circumstances, such as prior commands. You may use the DT utility to create a file containing 80 byte EBCDIC tape records with the command **'DT CONFIG.SYS TAPE:/CR/LF/T'**, which will be suitable for use with the above examples.)

Having tape data available from an operating system device opens a vast array of opportunities for transferring data. To realize these opportunities, however, you must be able to control the tape drive and I/O software to provide you with useable information. (For example, the above case required 80-byte EBCDIC records. Your data may not be in this form). The TDS software is controlled by a little known, but nevertheless useful, facility of the PC-DOS and MS-DOS operating systems, known as IOCTL.

The IOCTL facility provides a channel of communication through the operating system between the application, system and/or utility software and actual devices. IOCTL provides, in essence, an alternate form of read and write commands that pass control information, instead of data, to and from a device.

Perhaps the most severe constraint in using the IOCTL facility is the general lack of support it has received from utility and language software. Not all languages directly support IOCTL, and some of those that do fail to document this support. For example, IBM BASIC (and GWBASIC) contains an IOCTL$(*n*) function, which returns IOCTL data from the device upon which file *n* resides, and an IOCTL# command, which is similar to the PRINT# command except that it sends IOCTL information to the device instead of data, yet most of these versions of BASIC provide no documentation whatsoever to support these commands. (This has been remedied in some versions of BASIC 3.0 documentation).

The program **IOCTL.COM** is provided on the Catamount distribution diskette to provide a mechanism for transmitting and receiving IOCTL information through the DOS command line. For most applications, the Catamount IOCTL program will provide an adequate method of communicating with the TDS software.

7.6.1: IOCTL Command Syntax

Command syntax for the IOCTL program is as follows:

IOCTL *dev* [*string*]

If the IOCTL program is used to send IOCTL information to and from a disk drive, a disk drive designator, followed by a colon, should be used for the *dev*. For a character device, the device name, optionally terminated with a colon, should be used.

The string optionally following the *dev* specification is the actual string of information to be sent to the device. All characters entered on the command line after the *dev* specification are sent to the device via the IOCTL facility.

After transmitting the string, the device is interrogated for a responding IOCTL *string*. If one is available, it is displayed upon the standard output device, normally the console.

Thus, the IOCTL program simply sends a command string to the specified device, fetches a returned status string from the device, if one is available, displays the status string on the standard output device and returns to DOS. The device driver itself, not the IOCTL program, actually processes the data that is exchanged.

IOCTL information may also be sent to the TDS software via the IOCTL system call, (documented in the DOS Technical Reference manual) either directly via an operating system call or indirectly through other software's support of the IOCTL facility, such as the BASIC IOCTL$ and IOCTL#.

An example of use of the IOCTL program to send a command string, in this case /BOT to rewind the tape drive, to the TDS software is as follows:

```
IOCTL  D: /BOT
```

or

```
IOCTL  TAPE0: /BOT
```

This rewind operation will respond with a 000 or 003 error code returned by the IOCTL program, indicating respectively a successful completion of the rewind or that the tape drive was offline. (*The TDS software returns a 5 character status string, consisting of three decimal digits followed by a carriage return and line feed, for each error it has encountered. In addition, whenever a request for IOCTL information is made to the TDS software, three zeros are returned if no errors have occured since the last IOCTL status*).

Twenty-five commands are provided to control tape I/O functions such as tape positioning, translation, blocking, etc.:

/R:*n* is used to specify the logical record length for both reading from and writing data to tape. If no value has been specified since the TDS software was loaded, a value of 80 bytes is assumed. *n* must be a decimal integer with a value of 1 to 65535, without commas.

/B:*n* is used to specify the physical block size to be used when writing data to tape. If this value has not been specified since the TDS software was loaded, a value of 80 bytes is assumed. *n* must be a decimal integer with a value of 1 to 65535 and must not be larger than the maximum block size established when the TDS software was loaded.

/U specifies that "unformatted" records are to be accepted. This removes the restriction that the physical block size be a multiple of the logical record length. This option is recommended where variable length tape records are used, or to transfer data where correct record blocking on tape need not be verified. When this option is specified, both the physical block size and the logical record length (as specified with the /B:*n* and the /R:*n* parameter, respectively), are reset to a value of 2048. These may be reset by following the /U parameter with their new values.

/XON specifies that translation of tape data is to be performed. Translation remains in effect until reinstated by the /XOFF command, and assumes that tape data is in EBCDIC and computer data is in ASCII.

/XOFF specifies that no translation is to be performed upon tape data unless translation is later requested. If the /XOFF command has not been specified since the TDS software was loaded, translation will be performed.

/RO specifies that the tape is to be treated as a read only tape, with the TDS software returning an error if attempts to write data to the tape or write file marks upon the tape are made.

/RW specifies that the tape may be both read from and written to, as long as a "write enable" ring is installed in the tape reel.

/BOT specifies that the tape is to be positioned at the physical beginning of tape by rewinding the tape.

/EOT specifies that the tape is to be positioned at the logical end of tape by skipping at high speed until two consecutive file marks are found. This may cause the tape drive to run off the end of the tape on an uninitialized tape or a tape which does not indicate the logical end of tape with a double file mark.

/UNL specifies that the tape drive is to be rewound and unloaded. This operation returns control immediately after issuing the appropriate instruction to the tape drive, without waiting for the unload sequence to be completed.

/SF:*n* specifies that the tape is to be positioned by skipping *n* file marks at high speed. This positioning will be aborted if a double file mark is encountered, signifying the logical end of tape.

/SB:*n* specifies that the tape is to be positioned by skipping *n* physical blocks. This positioning will be aborted if a file mark is encountered. If a block has already been partially read, the remainder of the block counts as a skipped block. This positioning is primarily intended for use in situations where the user's software keeps track of physical blocks. As "short blocks" may occur within files generated by other operating systems (and this software as well) if a file is extended, the /SR operation should be used when it is important to skip an integral number of records.

/SR:*n* specifies that the tape is to be positioned by skipping *n* logical records. If a partial record has been returned to a prior read request, the remainder of the record counts as a skipped record. This operation is not valid if a buffered write operation is pending, and will be aborted if a file mark is encountered.

/SC:*n* specifies that the tape is to be positioned by skipping *n* bytes from the current tape position. If the current buffer is emptied a new buffer will be read until the specified number of bytes have been skipped. This operation is not valid if a buffered write operation is pending, and will be aborted if a file mark is encountered.

/RSF:*n* specifies that the tape is to be positioned by skipping *n* file marks in the reverse direction at high speed. This positioning will be aborted if a file mark or the beginning of tape is encountered.

/RSB:*n* specifies that the tape is to be positioned by skipping *n* physical blocks in the reverse direction. If a block has already been partially read, the preceeding portion of the block is counted as one block. This positioning will be aborted if a file mark or the beginning of tape is encountered. As with the /SB operation, this operation is not recommended for skipping over an integral number of logical records.

/RSR:*n* specifies that the tape is to be positioned by skipping *n* logical records in the reverse direction. If a partial record has been returned to a prior read request, the preceeding portion of the record is counted as one record. This positioning will be aborted if a file mark or the beginning of tape is encountered.

/RSC:*n* specifies that the tape is to be positioned by skipping *n* bytes from the current tape position in a reverse direction. If there are not enough bytes in the current buffer to accomodate this operation, the tape will be physically backspaced twice and the prior buffer will then be read in a forward direction, and this positioning will continue until the specified number of bytes have been skipped. This operation will be aborted if a file mark or the beginning of tape is encountered.

/WFM specifies that any data remaining in the tape buffer is to be written to tape, followed by two file marks. The tape is then backspaced over one file mark to position it for writing another file.

/CR specifies that trailing blanks are to be deleted from records when read, and appended to records when written to pad the records to the appropriate logical record length. Data records read from tape will be delimited with an ASCII carriage return character when this command has been specified. When this command has been specified and data is written to tape, carriage returns are recognized as record separators and discarded without being written upon the tape.

/LF specifies that line feed characters following carriage returns are to be ignored and not written to tape when the /CR command has been specified when writing to tape. Line feed characters will be appended to data records read from tape when this command and the /CR command have been specified.

/T specifies that tab characters to be written to tape are to be expanded with blanks to allow the next character to reside in the next position after a position divisible by 8. This is useful when using utilities such as COPY to create tapes from text files. When /T has been specified for files read from tape, a tab character will replace spaces immediately preceeding each position within a record that is divisible by 8.

/NT specifies that tab expansion selected by /T is to be disabled and remain disabled until reselected by a /T.

/RESET resets all internal parameters within the TDS software for the selected tape drive to their original, default values. After a /RESET has been processed, translation is in effect, any buffered data is discarded, tab expansion is disabled and the tape is enabled for both reading and writing.

/PURGE causes all outstanding buffers to be purged, either by writing them to tape if output is in process, or by discarding them on input operations. Once a purge operation has taken place, subsequent files on the *pseudo disk* may be written or read from the beginning as if a close had taken place.

As an example, to switch the TDS software to the unformatted record mode, the IOCTL program could be used together with the following TDS commands:

```
IOCTL  D: /U
```

From BASIC, the following program lines would select carriage return and line feed processing for tape data records and then display the resulting error code:

```
10 OPEN "TAPE0" FOR OUTPUT AS #1
20 IOCTL#1,"/CR/LF"
30 PRINT IOCTL$(1)
40 CLOSE 1
```

If TDS errors occur while performing I/O, the TDS device driver will post an error 0CH, General Device Failure, to the operating system. The operating system, upon being notified of the error, will then print a message upon the console (the contents of which varies from one release of the operating system to the next, but resembles the error given. For example, when you try to read a diskette without a diskette in the slot) and prompt the operator for a response indicating whether the operation should be aborted, retried or ignored. In most cases, the operator should abort and then use the IOCTL command or facility to fetch the error code from the TDS software and take whatever corrective action is necessary.

7.6.2: IOCTL Error Codes

Error codes may be fetched by your software through the IOCTL function. The IOCTL program requests and displays outstanding TDS errors each time it is executed.

Error codes that may be returned as a result of errors encountered when processing TDS IOCTL commands or performing TDS I/O operations are:

Error	Description
000	No error occured on the last operation or there are no outstanding errors to be reported.
001	A file mark was encountered while reading or positioning tape. (When reading, this is a normal end-of-file).
002	A DMA Overflow condition occured.
003	The tape drive was found to be in an offline condition.
004	A non-correctable read error was encountered.
005	A correctable read error occured and the data was returned sucessfully.
006	The beginning of tape was encountered when positioning the tape.
007	Internal software error.
008	A tape operation that was supposed to transfer data failed to transfer any data before completion.
011	A non-correctable interface error was detected.
012	A physical end of tape was encountered by the prior operation or a positioning command encountered a logical end of tape.
013	The physical beginning of tape was encountered on a reverse operation.
014	A block found on tape exceeded the maximum block size allowed when the **TDS** was installed.
015	A request has been made to write upon a tape considered to be a read only (RO) tape or that did not contain a write enable ring.
020	The current record length and block size are incompatible.
021	A logical sector was requested out of sequence during a read operation, indicating that random access was attempted on the TAPE *pseudo disk* file.
022	A logical sector was written out of sequece, indicating that random access writes were attempted on the TAPE *pseudo disk*.
023	An attempt was made to do a purge operation while the device was still open.
252	An attempt has been made to write a second file to the directory.
253	A invalid or zero value has been specified for an operation.
254	An uninterpretable value has been specified.
255	A request has been made that is uninterpretable or improper.

Error codes are buffered by the TDS software. Approximately 40 error codes may be retained by the software before an overflow occurs. All outstanding error codes are returned whenever an IOCTL request is made to the TDS software. If no errors have occured or are outstanding in the buffer, 000 is reported.

Whenever the TDS software determines that an output file on a TDS *pseudo disk* has been closed, all buffered data is written to tape and then two file marks are written to tape, the tape is then backspaced over one of these file marks, the software sets a flag indicating a media change to the operating system and reinitializes itself to its original condition, with the exception of the letter "v" in the volume name being toggled between a capitol "V" and a lower case "v". (This is done to alter the "label name" of the "removable media"). The *pseudo disk* is then available for reuse, and may be used to write additional files to tape. The software determines that the file has been closed by noting that an updated value for the file length has been posted to the directory.

The /WFM command should be issued to the TDS software to close output files on the TDS character devices (TAPE0: through TAPE7:). The /PURGE comand may be used to close read files, or to terminate write file buffering without writing file marks.

7.7: TDS Software Compatibility

It is important to realize that the TDS software does not provide an all-inclusive, full emulation of a disk drive and/or character device, as such emulation would not be useful for the purpose of transferring data to and from interchange tapes. Disk drives, as seen by the operating system, must have a structure the same as that defined by the operating system, i.e., 512 byte sectors, random addressability, standard directories, file allocation tables, bad track maps, etc. The structure of tape interchange data, however, is normally that used by traditional tape drive systems: a sequential, unit record device utilizing file marks, label records, record blocking, a broad variety of record lengths, and so forth.

The purpose of the TDS software is more to provide a vehicle for spanning the gap between these two very different protocols than to add an additional operating system device.

In order to make this vehicle possible, it is necessary to limit the form of the structures on either end, to structures that have use and meaning on both ends. The TDS software does this by providing the ability to read or write a single file (at a time), in sequential order, from the beginning of the file to the end of the file, using a single open, a series of sequential read or write calls, followed by a single close. Other sequences or methods of access will not work with this software and may cause errors or very strange results. Furthermore, not all utility, system and/or language software permits access to files in this manner, but instead makes the assumption that any block device attached to a computer using the DOS or MSDOS operating system is a full functioning, random access disk type device, even where only sequential access is required. Software making this assumption may very well not be able to transfer data to and/or from tape via the TDS software.

Other difficulties present themselves when attempting to perform this transcription of two different data media. With disk drives, because access is random, end-of-file information is necessary and inherent in the maintenance of directories and can be accessed at any time by the software. No similar mechanism is provided by standard 1/2" tape formats. An end-of-file situation on 1/2" tape is simply marked with a file mark, of which its presence is not known until it is actually encountered. DOS, being designed to operate only with directorized disks, assumes that it always knows where the end of file occurs, and does not even have a return status of "end-of-file" available to device drivers. Various assumptions have to be made to handle end-of-file situations, and methods valid for some applications are not valid for others.

Furthermore, although the situation is continuously improving in this respect as more mature versions of the operating system appear, there still are differences between how character devices and block devices (disks) are handled by the operating system. Also, because of some significant weaknesses in the original version of the DOS operating system, many system language and utility programs circumvent the operating system and attempt to unnecessarily manipulate and control access to various I/O functions themselves. Perhaps the only way to tell whether or not a particular software configuration is compatible with the TDS software is to try it.

A good example of the considerations required to use the TDS software is its use through BASIC. IBM BASIC remains virtually unchanged through three major releases of the DOS operating system. As IBM BASIC existed under DOS 1.0, it was capable of functioning in an operating system environment that provides only the most primitive device management. BASIC, therefore, was developed to provide its own management of I/O devices. Due to BASIC's attemt to fetch directory information from any file name it does not recognize as one of its own standard character devices, character devices not specifically integrated into BASIC by IBM and Microsoft simply do not function. BASIC attempts to fetch this directory information for all files it opens through the operating system to determine the length of the file, among other things. Character devices defined under the operating system, however, do not have randomly accessible directories.

(BASIC even goes so far as to provide its own host of interrupt service routines for devices it supports, whether or not they actually exist on the system. Interrupt vectors installed to support other devices prior to executing BASIC are simply destroyed upon entering BASIC).

Not only does BASIC exclude the use of operating system character devices, it also excludes the use of the IOCTL function to communicate with disk devices. This is apparently due to a weakness in the operating system that uses different types of calls for character and disk devices. To function correctly with both, software must first interrogate the device for its type, and then issue the appropriate call. BASIC does not do this. Therefore, in order to use the TDS software from BASIC, it is necessary to issue IOCTL commands to the character device, TAPE0:, yet pass data to and from the disk device and file, D:TAPE.

Another weakness of both BASIC and other DOS 1 compatible utilities and software is that communication to and from character devices is normally performed a single character at a time. The DOS COPY and TYPE commands continue to operate in this fashion, and prove to be excruciatingly slow when used with the TDS TAPE0: device (or any other character device, for that matter). Although this "single character at a time" bottleneck was eliminated with the advent of DOS 2, most software has not been modified to take advantage of the improved I/O transfer capability of DOS 2 and subsequent versions.

In general, the character device structure of DOS 2 provides what could have been an excellent interface to 1/2" tape and other devices. It is the actual implementation of system, application and utility software that has prevented this from being realized. This is almost exclusively due to the weakness of DOS 1 and the understandable reluctance of the software suppliers to restructure their software and abandon compatibility with earlier releases.

7.8: TDS Disk-Type Device Data Transfer

Eight block type devices (seen by the user as disk drives) are defined to the operating system by the **TDS.DEV** primitive device driver. The **TDS.COM** main device driver then implements one or more of these *pseudo disks* as an image, or emulation of disk data representing data that is actually located on 1/2" tape.

Initially, the directory of each of these *pseudo disks* contains a single file named "TAPE", which consumes the entire logical disk drive. When this file is opened and read, the data that is returned is that actually read from tape.

To write data to tape, the "TAPE" file may be erased, which merely clears the directory, and subsequent data written to the file that replaces it (the first file in the directory) is actually recorded on tape. When this file is closed, any buffered data is also written to tape.

(The TDS software determines that an output file on a *pseudo disk* has been closed by testing for changes in the directory whenever the directory sector is accessed. DOS 2 does not notify a device of file openings and closings, as DOS 3 does).

If a file mark is encountered by the TDS software when reading tape data, the remainder of the current sector is filled out with ASCII Control-Z characters, recognized by some software to indicate an end of file. An error will then be returned to the requesting software if an attempt is made by the operating system to read the next sector of data. Remember that the length of the file "TAPE" is 32M bytes long, and the operating system will expect to be able to read 32M bytes of data before the end of the file. The software reading the TDS data must either know when to stop reading the file (presumably from the content or count of tape data) or be able to recognize the Control-Z characters to avoid encountering a DOS error.

On write operations, the actual length of data to be written to tape is obtained by the TDS software from the updated directory resulting from the close. (Disk data is only written in sectors of 512 bytes. It is rare that the actual file length is a multiple of 512. The logical length of the file, showing how much of the last block is actually used, is contained in the directory). After writing the last data to tape when the TAPE file has been closed, the TDS software then writes two file marks to tape and repositions the tape by backspacing over one of the file marks. This makes it possible to add additional files to the tape.

After reading or writing a file to or from tape, it is recommended that the TDS software be reinitialized, via the /RESET command through the IOCTL function. This will clear the end of file condition, restore the *pseudo disk* directory to its original condition, and return all parameters to their default values. The TDS software may then again be set up with various commands through the IOCTL function, and tape operations resumed.

Writing additional files to the TDS *pseudo disk* (other than erasing and writing files to the first position in the directory occupied by TAPE, unless erased and rewritten or renamed) will result in errors. The TDS *pseudo disk*, otherwise, is a memory based "virtual disk", similar to that defined by the popular VDISK program. Disk commands such as ERASE and RENAME will all function with the *pseudo disk*.

7.8: (...continued)

A caution is in order regarding the use of the TDS software with DOS utilities. The DOS COPY and TYPE utilities do not correctly recognize the Control-Z character as an end of file indicator in all cases. For the COPY program, Control-Z terminators will only be recognized if they occur within the first 64K bytes of data within a file, even if the /A parameter is specified. Furthermore, these programs do not permit data transfers of more than 64K bytes to or from character type devices.

7.9: TDS Character-Type Device Data Transfer

Along with the eight disk-type devices defined by the TDS software, eight character devices, TAPE0: through TAPE7: are also defined. (*Do not confuse these devices with the TAPE[n]: specifications used in other Catamount utility software. Although they are designated in the same fashion to promote consistency,* ***the Catamount utility software does not utilize the TDS software in any way***).

TAPE0: is entirely synonymous with the first TDS *pseudo disk* installed on the system, TAPE1: with the second, and so forth. Not only do these designations reference the same physical device, they reference the exact same logical device. If, for example, you set the physical blocksize of disk D: to 8192 with the following command:

```
IOCTL  D: /B:8192
```

the physical blocksize of data written to TAPE0: as well will be 8192.

Thus, IOCTL operations may be directed to either the character device for the first tape drive or the disk device, independent of which of these devices actually receive the corresponding I/O data.

The character devices, TAPE0: throught TAPE7:, are simpler and easier to use than the *pseudo disks*. They may simply be opened, read or written, and closed. The TDS software returns Control-Z characters when it encounters a file mark reading data, and continues to return Control-Z characters until either a command is received throught the IOCTL function or until the file is closed and then reopened, as recognized by the input flush system call. When writing data to tape, buffered data is written and the tape is closed whenever the operating system issues an output flush call. If it is necessary to close a tape file and begin a new one without the operating system flushing the device, the tape file may be closed by issuing the /WFM command through the IOCTL function.

7.10: Other Restrictions and Cautions

When using the *pseudo disk device(s)*, it is generally not possible to take advantage of the TDS commands that reposition the tape and alter the translation modes during the time the file is being accessed. Because requests are only made to block type devices on a sector by sector basis, and buffering is performed by both the TDS software and the operating system, commands issued by the IOCTL function may not appear to take place until data to or from the next sector, or even next several sectors have been processed. This may render the block devices unsuitable in applications where it is necessary, for example, to switch translation mode or reposition within a file. In addition, the block devices will not allow processing of files larger than 32 megabytes (33.5 million bytes).

These device service routines are not reentrant and will not function correctly if used in a full multitasking environment. (Some DOS 2.x routines, such as Set Interrupt Vector, also are not reentrant and do not operate correctly when called recursively). Should a user require the TDS routines for use with a reentrant multitasking operating system, "locks" must be installed in the TDS.DEV routine to force serial execution of the TDS software in order to maintain software integrity and to conform to the hardware constraint that limits each interface to a single operation at a time.

With version 3.0 and 3.1 of this software, DOS 3.0 and 3.1 will incorrectly interpret the BIOS Parameter Block (BPB) information presented by the TDS software when a cluster size multiplier of 1 is specified in the TCONFIG.SYS file, causing the operating system to return incorrect values for available space when directories are performed on TDS *pseudo disks*. This problem seems to be of no consequence. DOS 2, however, handles the BPB correctly, and a value of one is recommended for use with DOS 2, as DOS 2 requires a cluster size multiplier to accomodate more than 16Mbytes of TDS block device data, while DOS 2 incorrectly handles larger cluster size multipliers in some cases. When DOS 3.0 and 3.1 are used, values of 0, 2 or 3 are recommended and seem to all function correctly. When DOS 2 or 2.1 is used, a value of 0 or 1 is recommended.

Some versions of GWBASIC seem to be unable to operate with interrupt levels INT2-INT7 modified. Thus, when using the TDS or BPS software (or other software using the DTIOS or the device interrupt vectors) together with GWBASIC, it is necessary to not use interrupts. (Disable interrupts by specifying an interrupt level of zero in the TCONFIG.SYS file. If you are using a non-buffered drive, high speed operations must also be inhibited, as they will cause DMA overflow errors if interrupts are not used). A separate TCONFIG.SYS file may be defined with high speed operations suppressed and no interrupts specified and placed in a directory from which the GWBASIC program is executed, thus only restricting these features when GWBASIC is in use, while leaving them available for use from other software.

Notes

Chapter: EIGHT

The BPS Software

8.1 Introduction

The Catamount Basic Programming Support (BPS) Software consists of a set of programs and subroutines used to provide access to 1/2" magnetic tape data for IBM Personal Computer BASIC interpreter programs through the use of the Catamount 1/2" Magnetic Tape Subsystem. (The BPS Software will also work with certain versions of GWBASIC).

The BPS software implements the USR function of BASIC to transfer tape data, commands and error codes between the user's program and the Catamount subsystem. This method provides the user with simple, efficient access to tape data in a context compatible with the BASIC programming language. In addition, the BPS software frees the user of chores normally

8.1: (...continued)

associated with tape processing such as record deblocking and data translation (ASCII/EBCDIC and EBCDIC/ASCII).

The BPS software is directly derived from and fully upward compatible with earlier versions of the Catamount BPS software on the Apple][and Commodore computers. This time-tested software was firmly established in use well before the emergence of the IBM Personal Computer.

Three entry points are provided to access the BPS software. A read data subroutine that returns string data from tape to BASIC, a write subroutine that writes string data to tape, and a control subroutine used to pass commands and parameters to the BPS software. (The read subroutine also supports a toggle mode for compatibility with earlier versions of BPS).

The five files used with the BPS software, **BPS.COM**, **BPS.BAS**, **BASIC.BAT**, **FREE.COM** (also used with the TDS software) and **N.DAT** are provided on the Catamount distribution diskette.

8.2: Compatibility

The BPS software directly utilizes machine language subroutines internal to BASIC. Some of these subroutines are formally documented and supported by IBM BASIC, while others are not. These subroutines are only available in versions of interpretive BASIC written by Microsoft, existing in ROM within the IBM PC and within RAM on loaded versions of GWBASIC. These subroutines must exist in the same form as they exist in IBM BASIC and perform exactly the same function. Although current versions of GWBASIC satisfy this requirement, there is no guarantee that future versions will continue to do so.

Because of the direct and intimate relationship occuring between BASIC and the computer hardware, the BPS software is less likely to operate correctly with so-called compatible computers than other Catamount software and hardware. It is also less likely to be compatible with other peripheral devices.

This software utilizes the interrupt vector area reserved for interrupts 60H through 64H to establish linkage between the BASIC program and the USR function. This area must be free for this use and undisturbed by any other software.

BASIC programs using this software must either assign and leave the SEG pointer pointing to zero, or reassign SEG via the DEF SEG command prior to each call to the BPS software.

The BPS software requires the use of at least one USR vector. In this manual, it will be assumed that USR, USR1 and USR2 are all available for use by the BPS software. If other elements of the user's program require other use of the USR function(s), the selectoion of the USR vectors must be made to accomodate these usages.

The BASIC.BAT file on the Catamount distribution diskette contains the commands necessary to conveniently implement the BPS software for a single tape drive with a maximum tape block size of 32,768 bytes. This batch file loads the BPS software, loads the BASICA program existing in the current search directory, and runs a BASIC program that sets up BASIC for use with BPS.

In most cases, the user only need transfer **BASIC.BAT**, **BPS.COM**, **BPS.BAT** and **N.DAT** to his disk, make sure that the BASICA.COM program and the **TCONFIG.SYS** file is in the search directory path (via the PATH command) and enter the command:

```
BASIC
```

(If there is a program in the current search directory named BASIC.COM or BASIC.EXE that appears earlier in the search sequence, either that program or **BASIC.BAT** will have to be renamed so that **BASIC.BAT** may be executed).

The **BASIC.BAT** file contains the following two DOS commands:

```
BPS <N.DAT
BASICA BPS/F:8/S:512
```

The first executes the **BPS.COM** program on the Catamount distribution diskette, piping the **N.DAT** file in as input in case it is required. The second line executes the user's BASICA.COM program (which must be available to the operating system for execution), specifying increased file and buffer capacity (not necessary for the BPS software, but usually required by applications programs) and specifying that the **BPS.BAS** program is to be loaded and executed by the BASIC interpreter.

The **BPS.COM** file is a program that installs itself in memory to provide the actual device support functions. **BPS.COM** is a DOS program that terminates with the DOS Terminate and Remain Resident system call so that it will remain in memory for subsequent use by the BASIC program. The **BPS.COM** program may be removed from memory by executing the **FREE.COM** program supplied on the Catamount distribution diskette.

8.3.1: BPS Command Syntax

The command syntax of the **BPS.COM** program is as follows:

BPS[/D:*n*][/M:*n*][/B:*n*][/XOFF][/RO][/S][/X:*filespec*]

Seven parameters are provided that may appear on the BPS command line:

/D:*n* is used to specify the address (0-7) of the highest addressed tape drive on the Catamount subsystem in situations where more than one tape drive is to be used. This drive will be selected until another drive is selected with the BPS Function 17 reference (described below). *n* must be an integer with a value of 0 to 7. If this parameter is not specified, a value of zero will be used. One buffer is allocated for each drive 0 through *n*.

/M:*n* is used to specify the maximum block size that will be used. If this parameter is not specified, a value of 32K (32768) will be used. *n* must be a decimal integer with a value of 1 to 65535. As many buffers as there are tape drives specified by the /D: parameter are allocated from DOS memory, with each buffer being the size specified by the /M: parameter. These buffers are allocated in a manner that guarantees that they do not span a 64K byte physical address boundary. This may result in allocating substantially more memory than the size of this program and the requested tape buffers.

/B:*n* is used to specify the physical block size to be used when writing data to tape. If this parameter is not specified, a value of 80 will be used. *n* must be a decimal integer with a value of 1 to 65535, and must not be larger than the maximum block size specified by the /M parameter. This parameter may be changed within a BASIC program through the use of BPS Function 12.

/XOFF specifies that no translation is to be performed upon tape data unless translation is later reinstated. If the /XOFF parameter is not specified, translation will be performed with the assumption that tape data is in EBCDIC and computer data is in ASCII.

/RO specifies that the tape is to be treated as a read only tape, with the BPS software returning an error code if attempts to write data to the tape or write file marks upon the tape are made. If the /RO parameter is not specified, tape data may be written as long as a "write enable" ring is installed in the tape reel.

/X:*filespec* allows the use of externally defined translation tables. These translation tables must be 512 bytes long, with the first 256 bytes containing the corresponding ASCII values to replace EBCDIC characters, and the next 256 bytes containing EBCDIC values to replace ASCII characters. The ASCII.ASM and EBCDIC.ASM source files used to create sample translation tables are included in this release. The external translation tables are specified by using a /X:*filespec* in the command line of the utility or installation program. The filespec contains a *filename* and optionally a *disk* drive designator, *path*, and/or *extent*.

The **BASIC.BAT** file is set up to pass parameters that appear on the command line to the **BPS.COM** program. Thus, to take advantage of the **BASIC.BAT** file and at the same time specify parameters for the **BPS.COM** file, you may simply follow the name BASIC with the same parameters that are valid for use with the **BPS.COM** program.

If the BPS software has already been loaded and an attempt is made to load it again, the BPS program will notify the operator and ask if a second image is to be installed. In most cases, this should not be necessary, as the previously installed image is adequate. If BPS must be reinstalled (to change maximum block size, for example), it is usually best to remove the prior installation with the FREE command before installing a new version, because if both versions are installed at the same time, the prior version continues to occupy memory while it is inaccessible for use. (*The* **N.DAT** *file included in the* **BASIC.BAT** *file is used to avoid reinstallations by answering* **N** *to the question of whether or not to install an additional version*).

Once the **BPS.COM** program has been loaded, the machine language subroutines providing the tape drive support are available for use by BASIC programs.

In order to use these subroutines, the BASIC interpreter must be told where to find them. This may be accomplished in any of several ways.

The **BPS.BAS** program invoked by the **BASIC.BAT** command stream contains the commands to do this from within a program. If the **BPS.BAS** program is not to be used, the commands appearing in the first line of the **BPS.BAS** program may be used in other programs to accomplish the necessary linkage.

The following direct command may also be entered (to the BASIC interpreter):

```
DEF SEG=0:DEF USR=&H180:DEF USR1=&H185:DEF USR2=&H18A
```

This informs the BASIC interpreter that the base segment register for external references is to contain a value of zero, and that three USR function references, USR, USR1 and USR2, are to be assigned to the BPS vectors 180H, 185H and 18AH. The BPS vector at 180H points to the subroutine used to read tape data. 185H contains a jump vector to the write tape data subroutine, while 18AH is vectored to the control subroutine.

(The remainder of this manual will assume that USR, USR1 and USR2 are assigned to these three vectors, respectively. To achieve compatibility with other software, the user may need to assign other USR functions to these entry points, or perhaps alternate assignments of a USR function. Cases where this reassignment is required should be very rare).

Once the segment register (SEG) and three user functions have been assigned, the **BPS** software is ready for use.

With a tape mounted, online, and ready for use, the operator could display the first 80 characters of EBCDIC data on tape by simply entering the command:

```
PRINT USR(80)
```

With the USR function assigned to the BPS read tape routine, it will return (and in this case display on the screen) the first 80 bytes of tape data.

In a program, the contents of an EBCDIC tape may be displayed on the screen with the following one-line program:

```
10 PRINT USR(255);:GOTO 10
```

This program simply prints the next 255 characters (the maximum number of characters that may occur in a BASIC string) and repeats itself. (It will continue to repeat until it is stopped with the Ctrl-Break keys).

As has been shown in the above examples, references to the read subroutine pass a numeric value as the argument of the USR function, which then returns as its value a BASIC string containing the data read from tape. The value passed as the argument is rounded and used as a count of the number of bytes requested by the BASIC program. A string of length equal to the requested length is returned if tape data is available and if enough bytes are available in the current block to satisfy the request. For example, if the USR function requests a string of 250 bytes while only 80 bytes are in a block, a string of 80 bytes will be returned by the BPS software. Requests may fetch all or part of the data in the tape buffer. If fewer bytes are requested than remain in the buffer, data that is left over after the transfer will be returned with the next USR request. Because the maximum length of a string in BASIC is 255 bytes, the maximum number of bytes that may be transferred in a single request is 255 bytes.

If an error is encountered while reading the tape, a zero-length string (equal to "" in BASIC) is returned. An error code describing the nature of the error is retained by the BPS software and its value may be obtained using the USR2(0) function described later.

String data returned by the USR function can be included in any context that is valid for string data within BASIC. The following examples all represent valid references to data read from tape within BASIC:

```
A$=USR(128)
IF USR(11)="THIS STRING" THEN 100
PRINT C$+USR(32)
Q$=MID$(USR(80),2,4)
```

Basic programs writing data to tape pass data as strings to the USR1 function. The USR1 function returns a numeric error code as its result.

To write a single block of 22 characters containing the EBCDIC text "This is a test record.", followed by two file marks, a program such as the following could be used. (Command code 8, as used with the USR2 function, writes a file mark and is described later):

```
10 A=USR1("This is a test record.")+USR2(8)+USR2(8)
20 IF A<>0 THEN PRINT "Error Encountered While Writing Tape"
```

Data is assembled into blocks before being written to tape. Normally, data is accumulated in the buffer until the number of bytes in the buffer is greater than or equal to the physical block size (as set by command code 12). If a single reference to the write data routine sends more data to tape than will fit within the specified physical block size, the excess data will be discarded and no error will be reported. (This is useful in situations where it is necessary to pad records, as excess padding characters at the end of a block will be trimmed).

The USR2 function (assigned to the BPS control subroutine) may be used to control tape operation. For example, command code 10, defined later, rewinds the tape to the beginning of tape and returns an error code. To rewind the tape, the operator could use the following direct command:

```
PRINT USR2(10)
```

The BASIC interpreter will respond by displaying the error code. (A zero will be displayed for a successful rewind. Error code three will be returned if the tape drive is offline or goes offline during the rewind).

As another example, the following program would display the contents of the first 1000 bytes of tape data, rewind the tape, and repeat until interrupted by a non-zero error code or the Control-Break keys:

```
10 FOR X=1 TO 1000:PRINT USR(1);: NEXT
20 IF USR2(10)=0 THEN 10
```

Various options are provided via the BPS control subroutine (USR2). These control output block size, translation, tape positioning (such as rewinding, as shown in the above example) and end-of-file operations. These options are selected by including a command code as the value of the argument supplied to the USR2 function.

Some of the commands, such as code 12 which is used to set the output physical block size, require an additional argument. These commands require that the next BPS reference following the command reference be a second reference passing the required value as the argument. For example, to set the output physical block size to 133 bytes per block, the following BASIC command could be used:

```
A=USR(12):B=USR(133)
```

The command codes that may be passed to the BPS control function (USR2) follow. Commands requiring additional arguments are indicated with an asterisk following the command code:

Command Code	Function Performed
0	No Operation - Returns Prior Error Code
1	Select Write Mode (for Compatibility Only)
2	Select Read Mode (for Compatibility Only)
3	Specify ASCII Tape Data, ASCII Computer Data
4	Specify EBCDIC Tape Data, ASCII Computer Data
5	(Same as 3, for Compatibility Only)
6	(Same as 4, for Compatibility Only)
7	Specify No Data Translation
8	Write a File Mark to Tape
9	Purge Outstanding Buffers (Write Short Block)
10	Rewind the Tape
11	Rewind the Tape, Unload upon Completion
12*	Specify Block Size
13	Select High Speed
14	Select Low Speed
15	Select High Density (Special Drive Option)
16	Select Low Density (Special Drive Option)
17*	Select Tape Drive

Command Code	Function Performed
18*	Skip Forward Record
19*	Skip Reverse Record
20	Skip Forward File
21	Skip Reverse File
22	Re-Initialize Software
23	Report Status
24	Execute Read/Write Test
25	Fetch Last Block Size

The control subroutine (USR2) used to implement these commands takes an argument (either a command code, or on a second reference where indicated by "*") containing a numeric value and returns a numeric value representing an error code. (The Report Status and Fetch Last Block Size commands return special values that are not actually error codes but instead contain requested information).

(Command codes 1, 2, 5 and 6 are not normally used with the IBM versions of the BPS software. These are preserved to maintain compatibility with Apple][and Commodore implementations of BPS. Under the Apple][and Commodore versions, special meaning is associated with USR function requests where the value of the argument passed is zero. To avoid these special meanings (and associated context switches), DO NOT use the USR function together with an argument with a value of zero.

Command code 0 simply returns the last error code encountered by the BPS software. This error code is cleared by all other function references, and is effectively set to zero if no error occurs. Code 0 is primarily intended to be used to fetch the error code associated with a prior read operation that returned a zero length string, indicating that an error had occured. (Write and control subroutine requests, USR1 and USR2, return error codes as the returned value of the function when the actual error occurs).

Error codes returned by the BPS software are as follows:

Error Code	Meaning
0	No Error Occured
1	File Mark Found (End-of-File)
2	DMA Overflow (Interface Error)
3	Tape Drive Not Ready
4	Non-Correctable Error
6	BOT Found by Skip File Reverse or Skip Record Reverse
7	Bad Write - DMA Not Complete
8	Interface Error - Zero Length Record
10	Lost Position Error - Error Occured While Positioning
11	Lost Command Error (Interface Error)
12	EOT Detected (Physical End of Tape)
13	BOT Encountered on Reverse Operation
14	Block Too Large

Error Code	Meaning
15	Invalid Mode Selected
16	Buffer Size Too Small for Test
17	Error in Block Length on Test
18	Error in Data on Test

Command codes 3, 4 and 7 select the three available translation modes. EBCDIC to ASCII translation on read operations and ASCII to EBCDIC translation on write operations will be provided by the BPS software unless it has been disabled by code 3 or 7 or by the /XOFF parameter when BPS was loaded. This translation may be reinstated at any time through the use of command code 4.

When ASCII to ASCII translation, as selected by command code 3, is performed the high order bit of data read from tape is stripped (set to zero) on reads and set (to one) on writes. This makes it possible to read and write ASCII tapes with the high order bits set, in order to be compatible with formats such as those frequently used by Control Data.

When it is necessary to write ASCII tapes with the high order bit zeroed (the more common method) no translation should be selected, as specified by code 7.

Command code 9, Purge Outstanding Buffers, causes any data that is buffered by write operations (i.e., data that has been sent to the BPS software to be written but awaits actual transfer to tape until the current physical block is full) to be physically written to tape. This will result in a block of data shorter than the physical block size being written to tape if there is write data in the buffer. On read operations, this command causes the contents of the current buffer to be discarded. (This would include any data in a block that has been partially read).

Certain other commands also cause a purge operation to take place before the remainder of the command is executed. Because the Write File Mark operation performs a purge prior to writing a file mark, and because file marks are normally written at the end of a file, the last block in a file will usually be a short block containing the remaining data when the file was "closed" by the Write File Mark command. This short block is the conventional method of ending a file, and is handled correctly by most operating systems as well as Catamount software.

Purges occur before commands 8-17, 20, 21 and 24 are executed.

It is recommended that a program which is not the first to read tape data after loading the **BPS.COM** perform a purge operation so that data left over in the buffer from prior reads do not interfere with subsequent reads. In most situations, a program reading tape can both accomplish the purge and start at a known position by rewinding the tape with command code 10. If the tape is already at the physical beginning of tape, this operation has no effect but to purge the read buffer.

The Rewind and Unload command, code 10, instructs the tape drive to perform a rewind and unload operation and then immediately returns control to the BASIC interpreter. If your application requires that the software wait until the rewind has been completed before it continues (to prompt the operator, for example), use command code 23, Report Status, to wait until the rewind is complete.

Command code 12, Specify Block Size, sets the physical block size to be used for writing tape data. Tape data is accumulated in the buffer until a block of this size is present, then an actual write operation takes place. (*A write operation can also take place as the result of a purge*). If a single call, the write data subroutine (USR1) passes more data to be written than will fit in the specified physical block, the remaining data is ignored. The physical block size specified through this command is not used on read operations. Read operations use the actual block size encountered when tape data is read.

As with the other commands that require a parameter value be passed in the subsequent USR2 reference, the Specify Block Size command toggles the USR2 function to intercept the value passed in the next reference and uses it, in this case, to indicate the subsequent physical block size. Because the first reference to USR2 for command code 12 always returns a value of zero, it is both readable and convenient to use an addition to join the two references. As an example:

```
30 IF USR2(12)+USR2(2048)<>0 THEN PRINT "Blocksize Error":STOP
```

would be a valid program step setting the block size to 2048.

The BPS software normally addresses the highest addressed tape drive installed on the controller by the BPS.COM program. Command code 17 provides a method of switching the BPS software to another drive, so that multiple tape drives may be used. All references to the BPS software in between references to command code 17 are directed to the tape drive specified in the prior reference to command code 17.

A program that reads 80-byte records from tape drive zero and writes them to tape drive one could read as follows:

```
10 A=USR2(17)+USR2(0)              :REM SELECT READ TAPE DRIVE ZERO
20 A$=USR(80)                      :REM FETCH DATA INTO A$
30 IF LEN(A$)<>80 THEN 80          :REM IF OUT OF DATA THEN QUIT
40 A=USR2(17)+USR2(1)              :REM SWITCH TO THE WRITE DRIVE
50 IF USR1(A$)=0 THEN 10           :REM WRITE DATA, CHECK ERROR AND LOOP
60 PRINT "Error Occured Writing Tape"
70 GOTO 100                        :REM EXIT BECAUSE OF ERROR
80 A=USR2(17)+USR2(1)              :REM SWITCH TO WRITE DRIVE
90 A=USR2(8)+USR2(8)               :REM WRITE TWO FILE MARKS TO SHOW END
100 END
```

Separate buffers and control tables are maintained for each tape drive if more than one tape drive is installed. Therefore, if you are reading ASCII data, for example, from one tape drive while reading EBCDIC from another, you do not need to switch translation each time you switch tape drives.

Command codes 18 and 19, Skip Forward Record and Skip Reverse Record, do not actually skip records (as the logical record size is unknown to the BPS software) but instead skip the specified number of characters. Because data within the current buffer is counted, these commands may not cause actual tape motion if the number of characters to be skipped can be accomodated by adjusting the pointers within the current buffer. When the tape actually has to be physically repositioned, The Skip Reverse Record command causes the tape to be backspaced twice and then read forward once to fetch data from the prior block. Long character counts may cause many, indeed hundreds or even thousands of blocks to be fetched by these commands. These commands are not valid during write operations.

Command codes 20 and 21, Skip Forward File and Skip Reverse File, position the tape in the indicated direction until a file mark has passed the read head. Note that Skip Forward File will leave the tape positioned just past a file mark, so that the next read will return the data that immediately follows the file mark. Skip Reverse File leaves the tape positioned immediately before a file mark (or at the physical beginning of tape if no file mark is encountered) so that a subsequent read attempt will return a file mark (again, unless the physical beginning of tape was encountered).

Command code 22, Re-Initialize Software, resets all pointers and parameters to the values they originally contained when the BPS software was loaded. Any buffering in process will be discarded by this command, as no purge is performed.

The Report Status command, code 23, returns a value that does not conform to the normal error code values. The status returned is a number whose binary components may be evaluated separately, such as with the AND function in BASIC. The binary components are:

Bit	Meaning
0	Tape Drive Not Ready (Offline, Busy or Rewinding)
1	Drive Has Gone Offline Since Last Operation or Status
2	Tape is Write Protected (No Write Ring)
3	Tape is Rewinding (Usually after Manual Rewind or Rewind and Unload
4	Last Operation Moved Tape Past Physical End-of-Tape

The Execute Read/Write Test function, command code 24, will perform a read and write test to verify the correct operation of the tape drive, interface and computer and return an error code upon completion, with a value of zero indicating that the test has been completed successfully without error. Because this command will write over data at the beginning of the tape, all data on the tape will effectively be scratched. ***DO NOT USE THIS COMMAND UNLESS YOU INTEND TO SCRATCH THE MOUNTED TAPE.***

Command code 25, Fetch Last Block Size, also does not return an error code, but instead returns a numeric value indicating the block size of the last tape transfer. The physical block size of an immediately preceeding read (or write, especially in the case of a purge) may be determined through this function.

8.4: (...continued)

It should be noted that the USR function reference may be included in any context in which a string value is appropriate. Likewise, the USR1 and USR2 references may be included any time a numeric value is permitted. The BPS software places not constraints on the context in which the USR functions may appear.

The BPS software returns the BASIC error "Overflow" if an unrecognized function code is passed to the USR2 function or an invalid value such as a block size of greater than 65535 is specified, a tape drive address greater than the maximum drive address is selected, or a record length of greater than 255 is requested.

The BPS software returns the BASIC "Type Mismatch Error" if a numeric argument is passed to the USR1 function, or a string argument is passed to the USR or USR2 function. Basic will return its own "Type Mismatch Error" if a variable or function receiving the value returned by the USR function does not allow a string type value or if a variable or function receiving the value returned by a USR1 or USR2 function does not allow an integer type value.

8.5: Examples in Programming

The following program could be used to read a file from tape and copy it to disk:

```
10 A=USR2(10)+USR2(22)          :REM REWIND AND REINITIALIZE
20 OPEN "SAMPLE.TXT" FOR OUTPUT AS #1
30 A$=USR(255)                  :REM FETCH MAX SIZE STRING
40 IF LEN(A$)=0 THEN 70         :REM CHECK FOR ERROR
50 PRINT#1,A$;
60 GOTO 30
70 A=USR2(0):IF A=1 THEN 100    :REM NORMAL EXIT IS FILE MARK
80 PRINT "Error Code";A;" Detected While Reading Tape"
90 GOTO 110
100 PRINT "Normal End of Transfer"
110 CLOSE 1
120 END
```

A program to read in a block of tape data at a time and display its contents and length could be written as follows. Note that the first line contains initialization information so that the **BPS.BAS** program need not be run first:

```
10 DEF SEG=0:DEF USR=&H180:DEF USR1=&H185:DEF USR2=&H18A
20 A$=USR(255)                    :REM FETCH FIRST PART OF DATA
30 IF LEN(A$)=0 THEN 160          :REM SKIP OUT IF ERROR
40 C=USR2(25)                     :REM COUNT OF BYTES IN BLOCK
50 PRINT A$;                      :REM PRINT DATA FETCHED
60 C=C-LEN(A$)                    :REM SUBTRACT FROM REMAINING
70 IF C=0 THEN 100                :REM SKIP IF DONE WITH BLOCK
80 A$=USR(255)                    :REM GET MORE DATA FROM BLOCK
90 GOTO 50                        :REM AND LOOP BACK
100 PRINT "Block Length =";USR2(25);" ";
110 PRINT "Strike Return to Continue, Q to Quit:";
120 INPUT Q$
130 PRINT
140 IF (Q$="Q") OR (Q$="q") GOTO 210
150 GOTO 20                       :REM START NEXT BLOCK
160 E=USR2(0)                     :REM FETCH ERROR CODE
170 IF E<>1 THEN 200              :REM SKIP IF NOT FILE MARK
180 PRINT "File Mark Encountered, "
190 GOTO 110                      :REM LOOP BACK IF FILE MARK
200 PRINT "Error Code";E;" Encountered While Reading Tape."
210 END
```

The following program converts text records to 80-byte, 20 record per block fixed block ASCII tape records padded with spaces.

```
10 OPEN "TEST.TXT" FOR INPUT AS #1
20 A=USR2(22)+USR2(10)+USR2(7)+USR2(12)+USR2(80*20)
30 REM RESET AND REWIND IN CASE OF PRIOR USAGE
40 REM TURN TRANSLATION OFF AND SET UP BLOCK SIZE
50 IF A<>0 THEN 170                    :REM EXIT IF ERROR
60 IF (USR2(23) AND 4)=0 THEN 90       :REM OK IF WRITE ENABLED
70 PRINT "Write Ring Missing From Tape - Unable to Write"
80 GOTO 190
90 P$="                          ":P$=P$+P$:REM 80 SPACES
100 IF EOF(1) THEN 190
110 LINE INPUT#1, A$                   :REM GET DATA FROM DISK
120 IF USR1(LEFT$(A$+P$,80))=0 THEN 100
130 REM LINE 120 PERFORMS JUST ABOUT ALL THE FUNCTIONS REQUIRED
140 REM TO PAD AND WRITE THE DATA. IT ADDS THE DATA STRING TO
150 REM THE PADDING STRING, TAKES THE LEFT 80 BYTES, WRITES
160 REM THEM TO TAPE, CHECKS THE RETURNED ERROR CODE AND LOOPS
170 PRINT "Error Encountered While Writing to Tape"
180 GOTO 2000
190 PRINT "Normal End of Transfer"
200 CLOSE 1
210 END
```

Notes

Chapter: NINE

The GPTR Software

9.1 General Description

These assembler routines provide access to tape drive function through DOS LINKable subroutine calls. To use the GPTR (General Purpose Tape Routines) software, the programmer must be familiar with linkage protocols used by his application software and those used by the GPTR routines.

The standard linkage is compatible with the Lattice C small model linkage conventions. Other conventions can be easily accomodated through the use of a "driver" routine. An example of a driver routine using 32-bit pointers and "FAR" linkage to link with Microsoft FORTRAN can be found in the **GPTR.FTN** assembly.

9.2: GPTR Linkage

To use these routines with the Lattice C small model calling conventions, push the 16-bit offsets of the parameters defined in the data segment onto the stack, rightmost parameter first, and call the NEAR routines listed below. (*This is automatically accomplished by the Lattice C compiler*).

It is recommended that applications other than Lattice C use the entry points separately described below, or modify the **GPTR.FTN** driver.

The GPTR software stands entirely separate from the operating system, and does not use the **TCONFIG.SYS** file for configuration information. The tape drive and controller configuration for these routines is handled in the file **GPTR.DEF**. To alter the controller base address, DMA channel, or interrupt level, the appropriate changes must be first made in the **GPTR.DEF** file, and the GPTR software must then be reassembled and relinked with the calling program.

The GPTR software is designed to perform on a logical record basis, as opposed to a physical block basis. If access to physical block information is required, calls can determine the block size by placing read requests for a size that exceeds the maximum block size used, and fetching back actual block size information from the TLENGTH field. Optionally, the block size may be determined by fetching the first byte of a physical block, for example, and then fetching the physical block size via the TAPEBLOCK call, specifying a value of zero for the BLOCKSIZE parameter.

The standard entry points defined for the Lattice C linkage are as follows. Translation, Blocking/Deblocking, Positioning and Initialization are provided. Error codes returned correspond to the BPS error codes.

Entry points are provided as follows:

Procedures	Parameters
TAPEINIT	SPACE, BUFFERLENGTH
TAPEBLOCK	BLOCKSIZE, ERRORCODE
TAPEREAD	TRECORD, TLENGTH, ERRORCODE
TAPEWRITE	TRECORD, TLENGTH, ERRORCODE
TAPEREWIND	ERRORCODE
TAPEUNLOAD	ERRORCODE
TAPEMARK	ERRORCODE
TAPETRANS	TRANSMODE, ERRORCODE
TAPEFSKIP	CHARCOUNT, ERRORCODE
TAPERSKIP	CHARCOUNT, ERRORCODE
TAPEFFILESKIP	ERRORCODE
TAPERFILESKIP	ERRORCODE
TAPECLOSE	ERRORCODE

The first eight characters of the procedure name are actually used to provide linkage symbols. This is to conform with requirements imposed by some languages.

9.3: The GPTR Procedures

The TAPEINIT procedure must be called prior to calling any of the other GPTR procedures. Its purpose is to initialize variables within the GPTR software and to specify to the GPTR software the location of an area of memory which may be used by the GPTR software as a buffer. The argument SPACE must be an area of length greater than or equal to the value of the 16-bit integer BUFFERLENGTH. The area SPACE must not be used for other purposes between the execution of the TAPEINIT procedure and the last execution of any other GPTR routine within a program. Because of hardware limitations within the IBM PC, DMA transfers may not take place across physical 64K memory boundaries. *If the area SPACE spans a boundary, the larger portion of SPACE not spanning the boundary will be used and the value of BUFFERLENGTH will be updated to reflect the maximum blocksize that may be written.* ***Note that this value will never be less than half the initial value of BUFFERLENGTH.*** Thus, a certain blocksize can always be accomodated by specifying a BUFFERLENGTH and SPACE twice as large as the largest blocksize to be used. It is recomended to always specify a BUFFERLENGTH and SPACE twice the needed size to make sure you get the area you need, unless you can create a SPACE based off of a physical 64K boundary, or if you are short on memory.

The TAPEBLOCK procedure is used to specify the maximum block size to be used when writing data to tape. It may also be used to interrogate the software to determine either the amount of data in the current block when writing tape data or the size of the last block read when reading tape data by specifying a value of zero for BLOCKSIZE, which will be then updated with this information.

Normally, blocks are entirely filled before actually being written to tape. *When setting the blocksize, the value specified in BLOCKSIZE must not exceed the value returned by BUFFERSIZE when the TAPEINIT routine was performed.* As with all GPTR routines except TAPEINIT, an error code, as described below, is returned.

The TAPEREAD procedure returns tape data from the tape drive into the area TRECORD. The number of bytes to be transferred is specified by the value of the integer TLENGTH. If fewer bytes are available in the current block than were requested or if an error occured, TLENGTH is adjusted after the procedure is performed to reflect the actual number of bytes transferred

The TAPEWRITE procedure moves the number of bytes specified by TLENGTH from the area TRECORD to the tape buffer. If the tape buffer is then filled, it is written to tape. If TLENGTH specifies that more characters are to be written than will fit into the current block, then the value of TLENGTH will be updated to reflect the actual number of bytes written. If an error occurs TLENGTH will be set to a value of zero.

The TAPEREWIND procedure positions the tape at the physical beginning of the tape by issuing a rewind command to the tape drive.

The TAPEUNLOAD procedure rewinds the tape and unloads it from the hub allowing the tape to be removed from the drive.

The TAPEMARK procedure writes an ANSI compatible tape mark on the tape.

The TAPETRANS procedure is used to specify to the GPTR software the translation mode to be used when transferring data to and from the tape buffer. The valid values of the integer TRANSMODE are as follows:

Value	Tape Data	Computer Data
0	ASCII	ASCII
1	EBCDIC	ASCII
2	ASCII	ASCII
3	EBCDIC	ASCII
4	NO TRANSLATION	NO TRANSLATION

A default value of "1" (EBCDIC tape data, ASCII computer data) is established by the TAPEINIT procedure.

The TAPEFSKIP and TAPERSKIP procedures position the tape and/or the position within the current buffer forward or backward, respectively, by the number of bytes specified in the integer CHARCOUNT. If a file mark, the beginning of tape, or the end of tape is encountered while performing this operation, the operation is discontinued. The TAPEFSKIP and TAPERSKIP routines may not be performed while write data is being buffered.

The TAPEFFILESKIP and TAPERFILESKIP routines moves the tape to the position after the next tape mark and to the position preceeding the prior tape mark, respectively.

The TAPECLOSE operation is used to instruct the GPTR software to close out any tape buffering in process. The effect of this procedure while writing tape is to cause any data remaining in the buffer to be physically written to tape. If tape data was in the process of being read from the buffer when a call to this procedure occurs, then the buffer is flushed, with data for the next read coming from the next physical block on tape.

The operation performed by the TAPECLOSE routine is referred to as a purge operation. A purge is performed by the TAPEREWIND, TAPEUNLOAD, TAPEMARK, TAPEFFILESKIP and TAPERFILESKIP operations prior to the performance of their normal function. This normally alleviates the programmer of the responsiblity to close out the tape, however, the TAPECLOSE may be used to force the writing of short blocks to tape or to advance the tape to the next physical block.

In all of the GPTR routines, integers are treated as unsigned, positive quantities with a range of values from zero to 65,535 (64K minus 1).

9.4: GPTR Error Codes

Error codes are returned by all GPTR procedures except TAPEINIT. An error code of zero indicates that an operation was entirely successful. For read and write operations, error codes of 5 and 14 indicate that an operation is presumed to be successful, although some error correction occured. The value returned to the TLENGTH variable should be checked for a zero value on read and write operations to determine whether or not the operation was successful. Error codes returned by this software are as follows:

Error Code	Meaning
0	No Error Occured
1	Tape Mark Found (End-of-File)
2	DMA Overflow (Interface Error)
3	Tape Drive Not Ready
4	Non-Correctable Error
5	Correctable Error
6	BOT Found by TAPERSKIP
7	Bad Write - DMA Not Complete (Interface Error)
8	Zero Length Record on Read
9	Block Too Large on Read
10	Lost Position Error - Error Occured While Positioning
11	Lost Command Error (Interface Error)
12	EOT Detected (Physical End-of-Tape)
13	Reverse Command Given at BOT
14	Trivial Block Detected
15	Invalid Mode Selected
255	Invalid Operand or Invalid Value

9.5: C Linkage Notes

Parameters are required to be sixteen bit pointers placed on the stack. The value of these pointers are the offsets in the current data segment of the actual string variables, arrays, counts or error codes.

The stack on entry is presumed to point to the return address for a NEAR return. Above the return address are the parameters. This software returns with a simple NEAR return and does not specify a constant value to add to the stack after return to pop the pointers to the parameters off the stack. The calling routine must therefore do this. (This is a fairly common convention, as the 8086 does not allow a return on a variable parameter count, while many languages use calls allowing variable parameter counts).

The linkage conventions used by this software are designed to be compatible with the Lattice C Compiler for the SMALL (*single data segment and single code segment*) model. These conventions may or may not be met by other compilers. The meaning of the *rightmost* parameter may also vary, as other compilers (*notoriously Pascal compilers*) cause the leftmost parameter to be pushed first.

9.5: (...continued)

In most cases, a simple driver program can be written to adapt these routines to the calling and/or linkage conventions of other languages.

The **GPTR.DEF** file defines all hardware dependent operations, and may be edited prior to reassembly to modify this software to a particular hardware configuration.

To reassemble this software, the Microsoft Macro Assembler or any other Microsoft compatible Macro Assembler is required.

To link this program to the application software or driver program, the LINK program is required, as is knowledge of the linkage conventions and operation.

9.6: 32-bit Pointers and "FAR" Linkage

In addition to the standard linkage, entry points and variable symbols are provided to facilitate the writing of *drivers* to adapt the linkage conventions to those of other software. The following entry points and offsets are defined as public symbols, and may be accessed as near values within the PROG segment in the PGROUP group:

Linkage Symbol	Symbol Type	Usage
TAPE_INIT	NEAR	Procedure to handle Initialization
SAVE_REGS	NEAR	Saves all register values on entry
RESTORE_REGS	NEAR	Restores all register values prior to exit
DISPATCH	NEAR	Dispatches to appropriate subroutine
ERROR_POINTER	DWORD	Pointer to error variable location
ERROR_OFFSET	WORD	Offset portion of ERROR_POINTER
ERROR_SEGMENT	WORD	Segment portion of ERROR_POINTER
COUNT_POINTER	DWORD	Pointer to count variable location
COUNT_OFFSET	WORD	Offset portion of COUNT_POINTER
COUNT_SEGMENT	WORD	Segment portion of COUNT_POINTER
ARRAY	WORD	Offset in DS segment of Read/Write array
PARAM_COUNT	BYTE	Number of parameters passed to GPTR
END_OF_GPTR	BYTE	Symbol for last byte of assembly

Entry to TAPE_INIT is as follows:

ES:DI points to first byte of memory area made available to the GPTR software for internal buffering. (SPACE) in the TAPEINIT procedure.

DS:SI points to the parameter containing the initial and returned length of the actual buffer (BUFFERLENGTH).

Entry to DISPATCH is as follows:

AL	Function
0	READ
1	WRITE
2	REWIND
3	UNLOAD
4	SET BLOCKSIZE
5	SET TRANSLATION
6	FORWARD SKIP CHARACTERS
7	REVERSE SKIP CHARACTERS
8	FORWARD SPACE FILE
9	REVERSE SPACE FILE
10	CLOSE
11	WRITE FILE MARK

AH must be set to 1, 2 or 3, corresponding to the number of parameters appropriate for each call. (This value is saved in PARAM_COUNT by GPTR).

For single parameter calls, the two words in ERROR_POINTER must be correctly filled out with the segment and offset of the error code location to be filled out.

For two parameter calls, the error code pointer must be filled out as for the single parameter call. In addition, the COUNT_POINTER must be filled out with the segment and offset of the count location to be filled out.

For three parameter calls, both ERROR_POINTER and COUNT_POINTER must be filled out, as well as the variable ARRAY, which must point to the first byte of the I/O array. The offset of the array must be in the **DS** register.

The **GPTR.FTN** assembly provides an example of a driver routine which uses these external linkage symbols to provide a 32-bit pointer, FAR linkage which implements the same functions as defined by this assembly for the Lattice C linkage model.

The **GPTR.MSC** assembly provides FAR Labels for use with most C compilers. This linkage interface defines FAR symbols for linkage, which in turn call NEAR subroutines appearing in the **GPTR.OBJ** module, and sets up appropriate parameter passing information.

This linkage uses a *righthanded* (right-most parameter pushed first) parameter addressing scheme, as used by the standard GPTR. Additionally, this linkage does not *clean up the stack* on return, this is left to be done by the calling program.

This module is entirely separate from the other GPTR software, and must be separately specified at link time to be included for use.

GPTR.MSO is the object (linkable) code of the **GPTR.MSC** software.

The subroutine entry points (far public symbols) defined by this module are as follows:

"Far" Symbol Name	Corresponding "Near" GPTR Symbol
_MTINIT	TAPEINIT
_MTBLOCK	TAPEBLOCK
_MTREAD	TAPEREAD
_MTWRITE	TAPEWRITE
_MTREW	TAPEREWIND
_MTUNLD	TAPEUNLOAD
_MTMARK	TAPEMARK
_MTXLATE	TAPETRANS
_MTSKIP	TAPESKIP
_MTRSKIP	TAPERSKIP
_MTFILE	TAPEFILESKIP
_MTRFILE	TAPERFILESKIP
_MTCLOSE	TAPECLOSE

Since the **GPTR.OBJ** and the **GPTR.MSO** are not set up using the *.model* command, If you compile your C code directly to an .OBJ File with the more recent versions of the Microsoft Macro Assembler an error L2002 may occur during a LINK. To avoid this problem, compile the C code to an .ASM File and assemble it in the same manner as the **GPTR.ASM** and the **GPTR.MSC** Files.

9.6.1: Passing Parameters to "FAR" Subroutines

Example of the stack and the parameters for **_MTREAD** and **_MTWRITE**

SP+10h	errorcode Segment
SP+0Eh	errorcode Offset
SP+0Ch	tlength Segment
SP+0Ah	tlength Offset
SP+08h	trecord Segment
SP+06h	trecord Offset
SP+04h	Return Segment
SP+02h	Return Offset
SP:	

Example of the stack and the parameters for **_MTSKIP** and **_MTRSKIP**

SP+0Ch	errorcode Segment
SP+0Ah	errorcode Offset
SP+08h	charcount Segment
SP+06h	charcount Offset
SP+04h	Return Segment
SP+02h	Return Offset
SP:	

Example of the stack and the parameters for **_MTREW**, **_MTUNLD**, **_MTMARK**, **_MTFILE**, **_MTRFILE** and **_MTCLOSE**

SP+08h	errorcode Segment
SP+06h	errorcode Offset
SP+04h	Return Segment
SP+02h	Return Offset
SP:	

9.6.2: Externals and Variables

Below is a list of procedure and variable names that should be included in your C code.

```
extern void far MTINIT( char far *, int far *);
extern void far MTBLOCK( int far *, int far * );
extern void far MTREAD( char far *, int far *, int far * );
extern void far MTWRITE( char far *, int far *, int far * );
extern void far MTREW( int far * );
extern void far MTUNLD( int far * );
extern void far MTMARK( int far * );
extern void far MTXLATE( int far *, int far *  );
extern void far MTSKIP( int far *, int far * );
extern void far MTRSKIP( int far *, int far * );
extern void far MTFILE( int far * );
extern void far MTRFILE( int far * );
extern void far MTCLOSE( int far * );

#define     maxsize         16384

/*
* maxsize if the largest blocksize that you will need
* to be able to read or write.
*
*/
```

```
char    space(maxsize * 2);          /* Tape Buffer */
char    trecord(maxsize);            /* Transfer buffer */
int     bufferlength = maxsize * 2;  /* Max buffer length */
int     blocksize;                   /* Block size for reads and writes */
int     errorcode;                   /* Returned Error code */
int     tlength;                     /* Length to read/write */
int     skipcnt;                     /* Skip count */
int     transmode;
```

9.6.3: Procedure Names and Order

You use procedures as with other C procedures by calling name and passing all the needed parameters as pointers. You must first set up each variable as needed. The MTINIT must be called first. Then the MTBLOCK and MTXLATE in either order. At this point any of the other procedures may be called. MTBLOCK and MTXLATE may be called as often as required, but attention must be made as to never ask for a blocksize larger then the returned bufferlength from MTINIT.

```
MTINIT( space, &bufferlength );
MTBLOCK( &blocksize, &errorcode );
MTXLATE( &transmode, &errorcode );
MTREAD( trecord, &tlength, &errorcode );
MTWRITE( trecord, &tlength, &errorcode );
MTREW( &errorcode );
MTUNLD( &errorcode );
MTMARK( &errorcode );
MTSKIP( &skipcnt, &errorcode );
MTRSKIP( &skipcnt, &errorcode );
MTFILE( &errorcode );
MTRFILE( &errorcode );
MTCLOSE( &errorcode );
```

Chapter: TEN

The Catamount Controller Hardware

An Overview of the Catamount ATC-8 and ATC-16 Controller Cards, Their port addressing, and the Selection of DMA and interrupts.

10.1 Introduction

This Chapter was designed to give the needed information to programmers who wish to write their own direct control software for one of the Catamount Controllers, instead of using Catamount Software such as the GPTR. This Chapter only explains the operation of the Catamount Controller Cards and does not attempt to describe DMA operations, tape drive operations nor bus operations. The I/O programmer must consult the appropriate PC's Hardware Reference Manual, the Intel 8237 DMA Controller Data Sheet and the appropriate tape drive interface manual for this information in order to use these controllers for data transfer.

The term Bit 0 in this discussion shall mean the least significant Bit, while Bit 7 shall mean the most significant Bit of

10.1: (...continued)

a byte, and Bit 15 shall mean the most significant Bit of a word. Note that this is the same standard as used by the IBM PC computer and its documentation, while opposite of the designations traditionally used for tape drive documentation.

All values in this chapter that end with a lower case 'h' are to be understood as Hexdecimal values. All values that end with a lower case 'b' are to be treated as Binary values, and all others should be viewed as Decimal values.

Each Catamount controller connects a 'microcomputer I/O bus' to an industry standard 'formatted ½ inch tape drive interface'. A round data cable attaches between a 62 pin "D" type subminiature connector on the back of the controller card and the dual 50 conductor card edge connectors on the tape formatter or formatted tape drive.

Each controller occupies hardware I/O addresses, selectable on any even 32-byte boundary in the I/O address space from 200h through 3FFh via four shorting plugs on the controller card.

10.2: The ATC-8

The Catamount ATC-8 is a buffered DMA controller that controls the tape drive formatter and data transfers either by Programmed I/O or DMA.

The ATC-8 may generate an interrupt on IRQ2 through IRQ7. The interrupt level is selectable by shorting plugs on the ATC-8 and from within software.

The ATC-8 uses one of the three byte-wide DMA channels available on the I/O channel. The DMA channel is also selected by shorting plugs and software.

10.2.1: I/O Addresses for the ATC-8

The ATC-8 occupies thirty-two I/O addresses, but only sixteen of the thirty-two are used. Thus, the ATC-8 can be viewed as two groups of sixteen I/O ports. One at the base address, and the next at the base address plus 16.

The port addresses for the ATC-8 are as follows:

Offset	Default	Description	Style
00h	220h	Command port	OUTPUT
01h	221h	Select port	OUTPUT
02h	222h	Bus Function Control port	OUTPUT
03h	223h	Status port	INPUT
04h	224h	Force Cycle	STROBE
05h	225h	Clear Interrupts	STROBE
06h	226h	Flags port	INPUT
07h	227h	Clear Flags	STROBE
08h	228h	Extended Status port	INPUT
09h	229h	Read Data port	INPUT
0Ah	22Ah	Clear Flags Only port	STROBE
0Bh	22Bh	Clear Terminal Count INT	STROBE
0Ch	22Ch	Write Data port	OUTPUT
0Dh	22Dh	Clear FIFO Only port	STROBE
0Eh	22Eh	AUX Read Data port	INPUT
0Fh	22Fh	AUX Extended Status	INPUT

- OUTPUT ports are used to write information to the ATC-8.
- INPUT ports are used to read information from the ATC-8.
- STROBE ports are used to reset conditions on the ATC-8. An OUTPUT will initiate the STROBE operation.

The following is a breakdown of each port and its corresponding Bits:

Command	*220h*		*OUTPUT*
Bit	**Value**	**Description**	**Signal**
7	80h	Initiate Command	IGO
6	40h	Threshold 0	ITH0
5	20h	High Speed Off/On	IHISP
4	10h	Erase	IERASE
3	08h	Edit	IEDIT
2	04h	Write File Mark	IWFM
1	02h	Read/Write	IWRT
0	01h	Forward/Reverse	IREV

Select	*221h*		*OUTPUT*
Bit	**Value**	**Description**	**Signal**
7	80h	Formatter Enable	IFEN
6	40h	Formatter Address	IFAD
5	20h	Drive Address 0	IATD0
4	10h	Drive Address 1	IATD1
3	08h	Read Inhibit/Enable	
2	04h	Rewind	IREW
1	02h	Rewind and Unload	IRWU
0	01h	'Check Character' Gate	CCG

Bus Function	*222h*		*OUTPUT*
Bit	**Value**	**Description**	**Signal**
7	80h	NC	
6	40h	NC	
5	20h	NC	
4	10h	NC	
3	08h	Interrupt Mode	
2	04h	Last Word	
1	02h	Interrupt Enable	
0	01h	DMA Enable	

Status	*223h*		*INPUT*
Bit	**Value**	**Description**	**Signal**
7	80h	Formatter Busy	IFBY
6	40h	Load Point	ILDP
5	20h	NRZI Tape Detected	INRZ
4	10h	Tape Drive Ready	IRDY
3	08h	Rewinding	IRWD
2	04h	File not Protected	IFPT
1	02h	Data Transfer Busy	IDBY
0	01h	High Speed Selected	ISPEED

Force Cycle 224h			STROBE
Bit	**Value**	**Description**	**Signal**
All	All	Force data to the tape drive	

Clear Interrupts 225h			STROBE
Bit	**Value**	**Description**	**Signal**
All	All	Clears Interrupts	

Flags 226h			INPUT
Bit	**Value**	**Description**	**Signal**
7	80h	Hard Error Detected	IHER
6	40h	File Mark Detected	IFMK
5	20h	PE ID Burst Detected	IDENT
4	10h	End Of Tape Detected	IEOT
3	08h	Interrupting Source	
2	04h	Error Being Corrected	ICER
1	02h	Drive Offline	
0	01h	DMA Overflow	

Clear Flags 227h			STROBE
Bit	**Value**	**Description**	**Signal**
All	All	Clear Flag Values	

Extended Status 228h			INPUT
Bit	**Value**	**Description**	**Signal**
7	80h	Memory Parity Error	
6	40h	Formatter Busy	IFBY
5	20h	DMA Requested	
4	10h	Data Busy	IDBY
3	08h	Data no longer Busy	
2	04h	Drive ONLINE	IONL
1	02h	FIFO not Empty	
0	01h	FIFO not Full	

Read Data	*229h*		*INPUT*
Bit	**Value**	**Description**	**Signal**
All	All	Data Byte From Drive	

Clear Flags Only	*22Ah*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear Flag Values	

Clear T/C Interrupt	*22Bh*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear Interrupt Latch from 'Terminal Count'	

Write Data	*22Ch*		*OUTPUT*
Bit	**Value**	**Description**	**Signal**
All	All	Data Byte To Drive	

Clear FIFO Only	*22Dh*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Reset and Clear FIFO	

AUX Read Data	*22Eh*		*INPUT*
Bit	**Value**	**Description**	**Signal**
All	All	Data Byte From Drive	

AUX Ext Status	*22Fh*		*INPUT*
Bit	**Value**	**Description**	**Signal**
7	80h	Memory Parity Error	
6	40h	Formatter Busy	IFBY
5	20h	DMA Requested	
4	10h	Data Busy	IDBY
3	08h	Data no longer Busy	
2	04h	Drive ONLINE	IONL
1	02h	FIFO not Empty	
0	01h	FIFO not Full	

10.2.2: The Command Port

The eight Bits of the command port are wired directly to signals on the formatter interface. The low order five Bits are wired to the five IGO strobed command Bits, with Bits 0-4 corresponding to IREV, IWRITE, IWFM, IEDIT and IERASE, respectively. Bit 5 of this port is wired to the IHISP speed select signal. Bit 6 is wired to the ITH0 signal, and is normally not used. Bit 7 is the IGO strobe.

This port is clocked and electrically inverted, yielding a high voltage for a zero and a low voltage for a one, to match the negative logic used on tape formatter interfaces.

To issue a command to the formatter, the processor should place the tape drive command on the lower seven Bits of this port together with a logical one on the high order Bit, the IGO Bit, to lower the IGO command strobe.

The processor should then raise the IGO signal by writing the same value to the lower seven Bits of the port but changing the high order Bit to a zero. This will send a command to the tape drive.

Some of the Tape Drive Commands are as follows:

Lower Seven Bits of Command Port		Tape Drive Command
00h	0000000b	Read Block (Low Speed)
20h	0100000b	Read Block (High Speed)
02h	0000010b	Write Block (Low Speed)
22h	0100010b	Write Block (High Speed)
06h	0000110b	Write File Mark
10h	0010000b	Skip Record Forward
11h	0010001b	Skip Record Reverse
04h	0000100b	Skip File Forward
05h	0000101b	Skip File Reverse
1Dh	0011101b	Select High Density
1Ch	0011100b	Select Low Density
01h	0000001b	Read Record Reverse
0Ah	0001010b	ReWrite Record (Edit Write)
16h	0010110b	Write Erase Gap (Fixed Length)
1Eh	0011110b	Security Erase
12h	0010010b	Delete Record (Variable Length)

The formatter will respond by cycling the IFBY and IDBY signals to indicate the acceptance and completion of these commands. The IDBY signal is used to generate interrupts upon completion of these commands.

10.2.3: The Select Port

The select port is primarily used to perform tape drive selection. This port is clocked and electrically inverted, yielding a high voltage for a zero and a low voltage for a one, to match the negative logic used on tape formatter interfaces.

- Bit 7 of this port is wired to the IFEN (Formatter Enable) signal on the tape drive, which must be asserted (driven with a value of 1) to enable the formatter. If this Bit is released by writing a zero to it, pending tape drive operations will be aborted and the formatter will be reset.

- Bits 6-4 are wired to the IFAD and ITAD0 and ITAD1 address lines, used to address the tape drive on the controller. Thus, values of 08h through 0Fh on the high order nibble of this port will select the eight possible tape drives attached to the controller and assert the formatter enable signal.

- Bit 3 is used to control the internal DMA direction on the controller and to mask off "read after write" data during write operations. This Bit must be programmed to be a one during read operations and a zero during write operations.

- Bit 2 is wired to the IREW (Rewind) command line and may be lowered (by writing a one to this Bit) and then raised (by writing a zero) to cause the tape drive to execute a rewind operation.

- Bit 1 is wired to the IRWU (Rewind and Unload) command line and may be lowered and then raised, in a fashion similar to the Rewind Bit, to cause the tape drive to perform a rewind and unload operation.

- Bit 0 is wired to both the ITH1 (Threshold 1) signal on the tape drive and the internal check character gate enable on the controller itself. If this Bit is set to 1, then check characters are masked off when read from an NRZI (800 BPI) tape drive. When this Bit is zero, check characters are included in the data transferred. For most operations this Bit should be set to one.

10.2.4: The Bus Function Port

This port has only four Bits defined.

- Bit 3 controls the interrupt mode. If this bit is 0 an interrupt is generated at the End of IDBY except on read operations. On reads this interrupt will be delayed until any pending DMA operations are

complete or the tape drive has been taken offline. If this bit is a one an interrupt will be generated whenever the End of IDBY has been detected or the DMA terminal count has been reached.

- Bit 2 determines what method is to be used for the 'Last Word' signal to the tape drive. When this Bit has a value of zero, the 'Terminal Count' from the DMA controller is used as 'Last Word'. With a value of one, the 'FIFO Empty' is used as 'Last Word'.

- Bit 1 controls whether or not the interrupts are enabled or disabled. When a value of zero is written to this Bit, the controller disables the interrupts. A value of one enables interrupts.

- Bit 0 of this port performs in the same fashion as Bit 1, except it controls whether DMA is enabled or disabled.

All of the Bits controlled by this port are automatically cleared by the processors power on clear line to the condition where they are not asserting.

10.2.5: The Drive Status Port

The drive status port contains eight Bits directly wired to status lines driven by the formatter. This port electrically inverts the signals so that the sense of the signals is restored to positive logic, i.e., a one indicates that the value is true.

- Bit 0 is wired to the tape drive status signal ISPEED (High Speed Selected)

- Bit 1 is wired to the tape drive status signal IDBY (Data Busy)

- Bit 2 is wired to the tape drive status signal IFPT (File Protected)

- Bit 3 is wired to the tape drive status signal IRWD (Rewinding)

- Bit 4 is wired to the tape drive status signal IRDY (Ready)

- Bit 5 is wired to the tape drive status signal INRZ (NRZI Mode Selected)

- Bit 6 is wired to the tape drive status signal ILDP (Tape at Load Point)

- Bit 7 is wired to the tape drive status signal IFBY (Formatter Busy).

10.2.6: The Force Cycle Strobe

This strobe is only to be used at the beginning of a write operation. An output to this address will send the first byte of data from the FIFO RAM to the tape drive. A write command then must be sent to the tape drive to start the write operation. The data must be available in the FIFO RAM before this address is strobed.

10.2.7: The Clear Interrupt Strobe

This strobe is cycled by an output operation to its I/O address and causes the interrupt flip flop on the controller to be reset. The interrupt flip flop, while wired to the interrupt request logic, may be read by reading the flags port.

10.2.8: The Flags Port

The status of eight R/S (Reset/Set) flip flops are available in the flag port, indicating whether or not cycling has occured on the signals to which they are wired. These Bits may be used to check whether or not file marks, ID bursts, errors or completion of commands has occurred.

These Bits (except for the interrupt Bit, which is cleared separately) are cleared by the clear flags strobe. A value of one for these Bits indicates that the corresponding signal has been cycled, while a value of zero indicates that no cycling has occured.

- Bits 2, 4, 5, 6 and 7 are the outputs of flip flops wired to the ICER (Correctable Error), IEOT (End of Tape), IIDENT (PE Identification Burst/Check Character Gate), IFMK (File Mark) and IHER (Hard Error) signals from the formatter.

- Bit 3 is the interrupt Bit, indicating that a trailing edge of the IDBY signal has been detected. Bit 3 is a clocked flip flop, and must be cleared after being found TRUE.

  Although this is referred to as the interrupt Bit, it does not necessarily cause a CPU interrupt, as the interrupt driver must be enabled via the bus function port, the interrupt level itself must be enabled and an shorting plug must be installed for an interrupt to take place. This Bit can be better thought of as a completion Bit.

When sampling this Bit, it is a good idea to recheck the IDBY signal in the status port, as low logic level noise can cause falsing of the interrupt flip flop. (Falsing normally is caused by ringing or reflections of the transition of the IDBY signal from the high voltage level to the low voltage level). If the IDBY signal is zero, then a falsing has occurred, and the strobe should be cycled to clear the false interrupt condition. The IDBY signal should then be rechecked to verify that the interrupt has not occurred in the interim before the program continues waiting for an interrupt.

- Bit 1 is wired to the inversion of the IONL (Online) signal. This Bit is set if the tape drive is taken offline, even if only momentarily.

- Bit 0 represents whether or not a DMA operation has been requested by the tape drive while one is still outstanding on the controller. This Bit is set to one if a DMA overflow condition occurs.

10.2.9: The Clear Flags Strobe

This strobe clears the FIFO and the bits in the flag port except for the interrupt bit.

10.2.10: The Extended Status Port

- Bit 7 indicates that a parity error has occurred reading the FIFO RAM. This would indicate that the ATC-8 RAM buffer would be at fault.

- Bit 6 is the real-time IFBY (Formatter busy). This Bit is not run through any kind of latch, so it gives an actual reading from the IFBY. A value of one would indicate that the formatter is busy, while a zero would indicate that the formatter is not busy.

- Bit 5 indicates that data is ready in the FIFO on a read operation, and that the FIFO is not full on a write operation. When this bit has a value of one, it is in a true state and a data transfer may be performed.

- Bit 4 is the real-time IDBY (Data busy). A value of one would indicate that the data is busy, while a value of zero indicates that data is not busy.

- Bit 3 goes to a true state of one when the IDBY (Data busy) signal goes inactive (when data is no longer busy), and if the 'End of Data Busy' interrupt is enabled, an interrupt occurs.

- Bit 2 is the real-time IONL (Online). When this Bit is set to one the drive is Online. If the Bit is zero, the drive is Offline.

- Bits 0 and 1 are used to read the status of the FIFO. Both of these have a true state of one, or a value of zero means the condition is not true. A value of one for Bit 1 indicates the FIFO is empty, and a value of one for Bit 0 indicates the FIFO is full.

10.2.11: The Read Data Port

This port is used to read a byte of data from the FIFO.

10.2.12: The Clear Flags Only Strobe

This port is used to clear flags without clearing data in the FIFO.

10.2.13: The Clear Terminal Count INT Strobe

This port is used to clear a Terminal Count Status as seen by the interrupt generation logic.

10.2.14: The Write Data Port

This port is used to write the next byte of data to the FIFO RAM. .

10.2.15: The Clear FIFO Only Strobe

.An output to this port will clear all the internal pointers in the FIFO. This needs to be done at the beginning of data transfers. The 'Clear Flags strobe' clears the FIFO and the flag bits. If you want to clear the FIFO only, this strobe should be used and not the 'Clear Flags strobe'.

10.2.16: The Auxiliary Ports

The Auxiliary ports are placed in the opposite order from the Read Data and the Extended Status Ports, to place the Data port as the Low byte and the Status as the High byte with a 16 Bit INPUT. This way Bit 1 of the Extended Status can be checked at the same time a read operation occurs, to see if there is any more data to be read.

The Bits for the Auxiliary Extended Status are the same as the Bits for the Extended Status which is addressed at the Base Address+08h.

The Auxiliary Read Data Port is the same as the Read Data Port which is addressed at the Base Address+09h.

10.3: The ATC-16

The Catamount ATC-16 is a 16 Bit buffered tape controller card, providing for data transfers either by Programmed I/O or by DMA.

Interrupt levels and DMA channel may be selected or deselected via software. The ATC-16 uses shorting plugs only to define what its base I/O address will be. This address must be the same in software and on the ATC-16.

10.3.1: I/O Addresses for the ATC-16

The ATC-16 uses thirty-two 8 Bit port addresses, or sixteen 16 Bit port addresses. Normally all software should access the ATC-16 with 16 Bit data transfers.

The ATC-16 ports are broken down as follows:

Offset	Default	Definition	Style
00h	220h	Command port	OUTPUT
02h	222h	Status port	INPUT
04h	224h	Write Data port	OUTPUT
06h	226h	Read Data port	INPUT
08h	228h	Configuration port	OUTPUT
0Ah	22Ah	Extended Status port	INPUT
0Ch	22Ch	Force Cycle	STROBE
0Eh	22Eh	Clear FIFO	STROBE
10h	230h	Clear Offline INT	STROBE
12h	232h	Read Tape Word Counter port	INPUT
14h	234h	Clear Terminal Count INT	STROBE
16h	236h	Clear Ready	STROBE
18h	238h	Clear Clock INT	STROBE
1Ah	23Ah	Clear End of IDBY INT	STROBE
1Ch	23Ch	Clear First Byte INT	STROBE
1Eh	23Eh	Clear Status Bits	STROBE

- OUTPUT ports are used to write information to the ATC-16.
- INPUT ports are used to read information from the ATC-16.
- STROBE ports are used to reset data on the ATC-16. An OUTPUT to one of these ports will initiate the strobe.

The following is a breakdown of each port and its corresponding Bits:

Command	*220h*		*OUTPUT*
Bit	**Value**	**Description**	**Signal**
0Fh	8000h	Formatter Enable	IFEN
0Eh	4000h	Formatter Address	IFAD
0Dh	2000h	Drive Address 0	IATD0
0Ch	1000h	Drive Address 1	IATD1
0Bh	0800h	Read Inhibit/Enable	
0Ah	0400h	Rewind	IREW
09h	0200h	Rewind and Unload	IRWU
08h	0100h	'Check Character' Gate	CCG
07h	0080h	Initiate Command	IGO
06h	0040h	Threshold 0	ITH0
05h	0020h	High Speed Off/On	IHISP

04h	0010h	Erase	IERASE
03h	0008h	Edit	IEDIT
02h	0004h	Write File Mark	IWFM
01h	0002h	Read/Write	IWRT
00h	0001h	Forward/Reverse	IREV

Status	*222h*		*INPUT*
Bit	**Value**	**Description**	**Signal**
0Fh	8000h	Hard Error Detected	IHER
0Eh	4000h	File Mark Detected	IFMK
0Dh	2000h	PE ID Burst Detected	IDENT
0Ch	1000h	End Of Tape Detected	IEOT
0Bh	0800h	Interrupting Source	
0Ah	0400h	Error Begin Corrected	ICER
09h	0200h	Drive Offline	
08h	0100h	DMA Overflow	
07h	0080h	Formatter Busy	
06h	0040h	Load Point	ILDP
05h	0020h	NRZI Tape Detected	INRZ
04h	0010h	Tape Drive Ready	IRDY
03h	0008h	Rewinding	IRWD
02h	0004h	File not Protected	IFPT
01h	0002h	Data Transfer Busy	
00h	0001h	High Speed Selected	ISPEED

Write Data	*224h*		*OUTPUT*
Bit	**Value**	**Description**	**Signal**
All	All	Write Data Word to Tape Drive	

Read Data	*226h*		*INPUT*
Bit	**Value**	**Description**	**Signal**
All	All	Read Data Word From Tape Drive	

Configure	*228h*		*OUTPUT*
Bit	**Value**	**Description**	**Signal**
0Fh	8000h	Enable cable	
0Eh	4000h	Enable First Byte INT	
0Dh	2000h	Enable Offline INT	
0Ch	1000h	Enable End of IDBY INT	
0Bh	0800h	Enable Clock INT	
0Ah	0400h	Enable T/C INT	
09h	0200h	Enable Seperate Interrupts	
08h	0100h	FIFO Empty as Last Word	
07h	0080h	Write Ends on Odd Byte	
06h	0040h	- MUST BE 0 -	
05h	0020h	Enable Interrupts	
04h	0010h	Interrupt Select Mask MSB	
03h	0008h	Interrupt Select Mask	
02h	0004h	Interrupt Select Mask LSB	
01h	0002h	DMA Channel Mask MSB	
00h	0001h	DMA Channel Mask LSB	

Extended Status	*22Ah*		*INPUT*
Bit	**Value**	**Description**	**Signal**
0Fh	8000h	Drive has gone Offline	
0Eh	4000h	First Byte Transferred	
0Dh	2000h	Drive has become Ready	
0Ch	1000h	Hard Error at Drive	
0Bh	0800h	Clock Count Wrapped	
0Ah	0400h	Terminal Count Occurred	
09h	0200h	Interface Parity Error	
08h	0100h	Last Byte was Odd	
07h	0080h	Memory Parity Error	
06h	0040h	Formatter Busy	IFBY
05h	0020h	DMA Requested	
04h	0010h	Data Busy	IDBY
03h	0008h	Data no longer Busy	
02h	0004h	Drive ONLINE	IONL
01h	0002h	FIFO not Empty	
00h	0001h	FIFO not Full	

Force Cycle	*22Ch*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Force data to and/or from the tape drive	

Clear FIFO	*22Eh*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear FIFO	

Clear Offline	*230h*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear Interrupt Latch from 'Drive Offline'	

Tape Word Count	*232h*		*INPUT*
Bit	**Value**	**Description**	**Signal**
All	All	Number Of 'Words' FIFO has Received	

Clear T/C INT.	*234h*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear Interrupt Latch from 'Terminal Count'	

Clear Ready	*236h*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear Interrupt Latch from 'Drive Ready'	

Clear Clock	*238h*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear Interrupt Latch from 'Clock'	

Clear End of IDBY	*23Ah*		*STROBE*
Bit	**Value**	**Description**	**Signal**
All	All	Clear Interrupt Latch from 'Data Busy'	

Clear First Byte 23Ch			STROBE
Bit	**Value**	**Description**	**Signal**
All	All	Clear Interrupt Latch from 'First Byte'	

Clear Status 23Eh			STROBE
Bit	**Value**	**Description**	**Signal**
All	All	Clear Status Bits	

10.3.2: The Command Port

The sixteen Bits of the Command port are wired directly to signals on the formatter interface, and to the ATC-16 control lines.

- Bit 15 is wired to the IFEN (Formatter Enable) signal on the tape drive, which must be asserted (Driven to a value of 1) to enable the formatter. If this Bit is released by writing a zero to it, the pending tape drive operations will be aborted and the formatter will be reset.

- Bits 14 to 12 are wired to the IFAD, ITAD0 and ITAD1 address lines, and are used to address the tape drive on the controller, thus a value of 08h through 0Fh on the high order nibble of this port will select the eight possible tape drives attached to the controller and assert the formatter enable signal.

- Bit 11 is used to control the internal DMA direction on the controller and to mask off 'read after write' data during write operations. This Bit must be set to a 1 (one) during a read operation and reset to a 0 (zero) during a write operation.

- Bit 10 is wired to the IREW (Rewind) command line and may be lowered (by writing a one to its Bit) then raised (by writing a zero) to cause the tape drive to perform a rewind operation.

- Bit 9 is wired to the IRWU (Rewind and Unload) command line and may be lowered and then raised, in a fashion similar to the rewind Bit, to cause the tape drive to perform a rewind and unload operation.

- Bit 8 is wired both to the ITH1 (Threshold 1) signal on the tape drive and the 'internal check-character gate enable' on the controller itself. If this Bit is set to 1 (one), then check characters are masked off when read from NRZI (800 BPI) tape drives. When this Bit is zero, check

characters are included in the data transferred. For most operations this Bit should be set to 1 (one).

- Bit 7 is wired to the IGO (Initiate Command) command line of the tape drive. To issue a command to the formatter, the processor should place the tape drive command in the Command port with this Bit set to a 1 (one). Then the processor should raise the IGO signal by rewriting the Command port with IGO reset to a 0 (zero). This will send the command to the formatter.

- Bit 6 is wired to the ITH0 signal and is normally not used.

- Bit 5 of this port is wired to the IHISP (high speed) select signal. When this Bit is set to a 1 (one), High speed is selected. If this Bit is reset to a 0 (zero), Low speed is selected.

- Bits 4 through 0 are connected directly to IERASE, IEDIT, IWMF, IWRITE and IREV, respectively.

Some of the commands that can be sent to the formatter are:

Lower Seven Bits of Comand Port		Tape Drive Command
00h	00000000b	Read Block (Low Speed)
20h	00100000b	Read Block (High Speed)
02h	00000010b	Write Block (Low Speed)
22h	00100010b	Write Block (High Speed)
06h	00000110b	Write File Mark
10h	00010000b	Skip Record Forward
11h	00010001b	Skip Record Reverse
04h	00000100b	Skip File Forward
05h	00000101b	Skip File Reverse
1Dh	00011101b	Select High Density
1Ch	00011100b	Select Low Density
01h	00000001b	Read Record Reverse
0Ah	00001010b	ReWrite Record (Edit Write)
16h	00010110b	Write Erase Gap (Fixed Length)
1Eh	00011110b	Security Erase
12h	00010010b	Delete Record (Variable Length)

The formatter will respond by cycling the IFBY and IDBY signals to indicate the acceptance and completion of these commands. The IDBY signal may be used to generate interrupts upon completion of these commands.

10.3.3: The Status Port

The sixteen status lines come either from the formatter or from the ATC-16 controller card. Most of the status lines pass through a flip flop and must be reset after each read to assure correct values.

- Bit 15 is wired to the IHER (Hard Error) signal from the formatter, and goes active (to a value of one) when a hard error occurs.
- Bit 14 is wired to the IFMK (File Mark) signal from the formatter, and goes active (to a value of one) when a file mark was detected during the last operation.
- Bit 13 is wired to the IIDENT (PE Identification Burst/Check character Gate) signal from the formatter, and is set to one when a PE ID Burst occurs.
- Bit 12 is wired to IEOT (End of Tape), and goes active when the EOT is detected.
- Bit 11 indicates that a trailing edge of the IDBY signal has been detected. Bit 11 is a clocked flip flop and must be cleared after each read operation to be accurate.
- Bit 10 going active (to a value of one) indicates that a correctable error has occured.
- Bit 9 is wired through a clocked flip flop to the inversion of the IONL (Online) signal, so that if the drive is taken offline, Bit 9 is set to 1 (one).
- Bit 8 represents whether or not a DMA operation has been requested by the tape drive while one is still outstanding on the controller. This Bit is set to a 1 (one) if a DMA overflow occurs.
- Bit 7 is wired to the IFBY (Formatter Busy) signal.
- Bit 6 is wired to the ILDP (Tape at Load Point) signal.
- Bit 5 is wired to the INRZ (NRZI Mode selected) signal.
- Bit 4 is wired to the IRDY (Drive Ready) signal.

- Bit 3 is wired to the IRWD (Rewinding) signal.
- Bit 2 is wired to the IFPT (File Protected) signal.
- Bit 1 is wired to the IDBY (Data Busy) signal.
- Bit 0 is wired to the ISPEED (High Speed Selected) signal.

All these Bits are active true, or they have a value of one when they are true.

10.3.4: The Write Data Port

This port is used to write a word of data to the FIFO memory. When the ATC-16 is in Programmed I/O mode, this port is used to write data to the FIFO RAM.

10.3.5: The Read Data Port

This port is used to read a word of data from the FIFO memory. When the ATC-16 is in Programmed I/O mode, this port is used to read data from the the FIFO RAM.

10.3.6: The Configuration Port

This port is used to send set-up information to the ATC-16 Controller. There are Bits to enable interrupt level, DMA channel, which interrupts will be used and to connect the ATC-16 to the tape drive formatter.

- Bit 15 is used to determine if the ATC-16 is connected to the formatter or not. If Bit 15 is reset to 0 (zero) the ATC-16 is disconnected from the cable, and no operation or command will be sent to the formatter. When this Bit is set to 1 (one) the ATC-16 connects and commands may be transmitted to the formatter.
- Bits 14 through 10 are a selection of interrupt possibilities, which may all be selected by writing a value of 1, or deselected by writing a value of 0 (zero). An interrupt is enabled if its corresponding Bit is set to 1 and if Bit 9 is set to 1.
- Bit 9 selects the type of interrupt method is to be used. If this Bit is set to 1 (one), then the interrupts selected through Bits 10 to 15 are enabled. If this Bit has a value of zero, then the only interrupt received is at the end of IDBY (Data Busy).

- Bit 8 is used to select which method will be used to determine 'end of block' or 'last word'. If Bit 8 is set to 1 (one), then the 'last word' is sent to the tape drive as the FIFO empties. If this Bit is reset to a 0 then terminal count from the DMA controller is used as 'last word'. For most operations this Bit should be set to 0.

- Bit 7 is set to a 1 (one) when there is an odd number of BYTES to be transferred. This is not for an odd number of WORDS. For example, you have 21 bytes to transfer, or 10½ words, leaving a byte remaining. The data is transferred in words until the last byte, which is then transferred to the drive as a byte.

- Bit 6 must be set to zero for future compatibility.

- Bits 5 through 2 control which interrupt level is to be affected by a pending interrupt. The following table shows which interrupt levels may be selected:

Bit 2	Bit 3	Bit 4	Bit 5	Interrupt Level
x	x	x	0	Interrupts disabled
0	0	0	1	Level 3
1	0	0	1	Level 4
0	1	0	1	Level 5
1	1	0	1	Level 6
0	0	1	1	Level 10
1	0	1	1	Level 11
0	1	1	1	Level 12
1	1	1	1	Level 15

  - 0 = Bit reset
  - 1 = Bit set
  - x = Bit doesn't matter.

- Bits 1 and 0 control which DMA request and acknowledge lines are to be used. The following table shows which channels may be selected:

Bit 0	Bit 1	DMA Channel
0	0	DMA disabled
1	0	Channel 5
0	1	Channel 6
1	1	Channel 7

10.3.7: The Extended Status

Several of the Bits in the extended status port are connected directly to the interrupt lines, thus when an interrupt is active, the corresponding Bit in the extended status is also true.

- Bit 15 is connected to the Offline INTERRUPT and should be checked before leaving the interrupt service routine. When the drive has been taken offline this Bit goes high until the Interrupt is cleared.
- Bit 14 is connected to the First Byte INTERRUPT. When the first byte has been transfered this Bit goes active, and will remain active until cleared.
- Bit 13 indicates that the tape drive has become ready from a non-ready state. This Bit is cleared by writing to the 'Clear READY' strobe.
- Bit 12 indicates that a hard error has occurred at the tape drive. The drive either could not read or write the data as requested.
- Bit 11 is set true each second, via the internal ATC one second timer. This Bit is also wired together with the Clock INTERRUPT, and must be checked before leaving the interrupt service routine. To clear this Bit, write to the 'Clear Clock INTERRUPT' strobe.
- Bit 10 indicates that terminal count from the DMA has occurred. This Bit is also wired to the Enable Terminal Count INTERRUPT, and must be checked before leaving the interrupt service routine. This Bit is cleared by writing to the 'Clear Terminal Count INTERRUPT' strobe.
- Bit 9 indicates that a parity error has occurred in the data being read from the Tape Drive. This Bit is cleared by writing to the 'Clear Flags' strobe.
- Bit 8 indicates that there is an odd byte waiting to be transferred into the FIFO buffer on a READ operation. If this Bit is set, the Force Cycle port needs to be strobed to finish the transfer.
- Bit 7 indicates that a parity error has occurred to the data being written to the FIFO, either reading from tape or writing from memory. This Bit is cleared by writing to the 'Clear Flags' strobe.

- Bit 6 is the real-time IFBY (Formatter Busy) signal, and is set to a value of 1 (one) when the formatter on the tape drive is busy.

- Bit 5 indicates when the FIFO is avaliable for data transfers. On a read, this Bit is set to 1 (one) when the FIFO has data available to be read. On a write, this Bit is set to 1 (one) when the FIFO is neither full nor busy.

- Bit 4 is the real-time IDBY (Data Busy) signal. A value of one would indicate that the tape drive data is busy, while a value of zero indicates that the tape drive data is not busy.

- Bit 3 goes to a true state of '1' when the IDBY signal goes inactive. (When data is no longer busy). And if interrupts are enabled, an interrupt occurs.

- Bit 2 is the real-time IONL (Online) signal. When this Bit is set to one the drive is Online. If this Bit is zero, the drive is offline.

- Bits 0 and 1 are used to read the status of the FIFO. Both of these have a true state of one. A value of one for Bit 1 indicates the FIFO is not Empty. A value of one for Bit 0 indicates the FIFO is not Full.

10.3.8: The Force Cycle Strobe

When everything is set up to start a data transfer TO TAPE, you need to send the first byte of data to the tape drive, by writing to this port 3 (three) times. This needs to be done three times due to the three-stage pipeline on the ATC-16. The data is moved one step at a time down the pipeline with each strobe. Due to timing constraints of the of the FIFO on faster machines, you will need to put a small delay between each strobe. Below is an example of this process in assembly code:

```
MOV   DX,022Ch    ; Force Cycle port
OUT   DX,AL       ; Strobe 1
OUT   84h,AL      ; Delay
OUT   DX,AL       ; Strobe 2
OUT   84h,AL      ; Delay
OUT   DX,AL       ; Strobe 3
OUT   84h,AL      ; Delay
```

On a read operation, if there was an odd byte transferred, you will need to force a Cycle. The last byte will not yet be in the FIFO RAM. The Force Cycle will move this byte along with a dummy upper byte into the FIFO RAM. It is recommended, that if you need to use the Extended Status after you force the dummy byte through, to save the Status and Extended Status values before you initiate this transfer, because they will have new values after the Force Cycle.

Below is an example of how to do this:

```
        MOV     DX,022Ah        ; Extended Status Port
        IN      AX,DX           ; Read the Extended Status

; Do all other checking and comparing for Interrupts
; and other errors here

        PUSH    AX              ; Save old Extended Status
        TEST    AX,0100h        ; Check for ODD BYTE
        JZ      SKIP_ODD        ; Skip if no ODD BYTE
        MOV     DX,022Ch        ; Force Cycle port
        OUT     DX,AL           ; Strobe
        OUT     84h,AL          ; Delay
        OUT     84h,AL          ; Delay
        OUT     84h,AL          ; Delay
        OUT     84h,AL          ; Delay
        MOV     DX,0232h        ; Read Word Count port
        IN      AX,DX           ; Read the Word Count
        ADD     AX,AX           ; Double it
        DEC     AX              ; Return it to and ODD value
        MOV     CX,AX           ; Store count in CX
        JMP     SKIP_CONTINUE   ; Go on with program
SKIP_ODD:
        MOV     DX,0232h        ; Read Word Count port
        IN      AX,DX           ; Read the Word Count
        ADD     AX,AX           ; Double it
        MOV     CX,AX           ; Store count in CX
SKIP_CONTINUE:
        POP     AX              ; Restore the old Extended Status
; The rest of you program goes here.
```

10.3.9: Clear FIFO

An output to this port will clear the FIFO flags, and reset its counters to zero, and will reset all the flag Bits.

10.3.10: Clear Offline Interrupt

An output to this port will clear the 'Offline interrupt valid' Bit in the Extended Status and the pending 'offline interrupt'.

10.3.11: Read Tape Word Counter

This 16 Bit input port contains the count for the number of whole WORDS read from the tape drive. If 21 bytes are transferred, only 20 bytes will transfer and the 'Odd byte' Bit will be set in the Extended Status. You must strobe the Force Cycle, then read this port. This port will then have a value of eleven for 11 words (22 bytes). Multiply this value by two to get the number of bytes. Knowing that the last byte was odd will tell you to decrement the count by 1, giving 21 bytes.

10.3.12: Clear Terminal Count Interrupt

An output to this port will clear the Terminal Count Bit in the Extended Status, and the pending Terminal count Interrupt.

10.3.13: Clear Ready

An output to this port clears the Ready Bit in the Extended Status.

10.3.14: Clear Clock Interrupt

An output to this port will clear the pending 'One second Interrupt' and the 'Clock Count Wrapped' Bit in the Extended Status.

10.3.15: Clear End of Data Busy Interrupt

An output to this port will clear the 'End of Data Busy' Bit in the Extended Status and the pending 'End of Data Busy' interrupt.

10.3.16: Clear First Byte Ready

An output to this port will clear the pending 'First Byte' Interrupt and the 'First Byte transferred' Bit in the Extended Status.

10.3.17: Clear Status Bits

An output to this port will clear the status flags and the parity error bits in the extended status.

Chapter: ELEVEN

Error Messages

11.1 General Information

When exceptional conditions are encountered, Catamount utility programs print a standard form error message consisting of the utility program name, an error code with a value of between 1 and 99 and an explanatory note.

The error code is also returned to the operating system as a return code, which may be subsequently tested through the use of '**IF ERRORLEVEL** *n*' batch command.

11.1: (...continued)

The errors may be classified into two basic groups.

Errors 1 - 31 and 99 are returned by the standard tape I/O routines (the DTIOS) and the standard library routines (DTLIB). These errors are common to all utility programs, and are returned when actual errors occur while performing tape operations or while parsing the command line or loading the TCONFIG.SYS file.

Error codes in the range of 32 - 98 are specific to the issuing program, and an error code in this range returned by one program has no necessary relationship to the same error code returned by a different program.

11.2: Common Error Codes

The error codes common to all programs and their potential causes are as follows:

Code	Explanation
1	Unexpected End of File Encountered -- A File Mark was encountered before completion of a requested operation, other than a Skip File operation. Example: **TPOS TAPE:/SB:100** When there are less then 100 Blocks to skip before a file mark is encountered.
2	Interface Error - DMA Overflow -- Drive Transfer rate set too high for the interface. -- DMA Channel conflict with another Device. -- Memory Managment Software can cause this error. ++ Disable the Memory Manager and retry your application. -- Memcache, on most 80286 and 80386 machines, can cause this error. ++ Disable the Memcache and retry your application. -- Hardware failure.
3	Tape Drive Not Ready -- Make sure that the Tape Drive is on.line. -- Make sure the cable is attached correctly both on the computer and the Tape Drive, check that no pins have been bent in the 62 pin connector on the computer side of the cable. -- Make sure the card is not installed in the rightmost slot on an XT. The slot next to the power supply on an IBM PC/XT does not function the same as the other slots. -- **ATC-16 USERS:** The interface card must be in a 16 Bit slot and not an 8 Bit Slot. -- There may be an I/O Address conflict with another device.

Code	Explanation
4	Non-Correctable Tape Data Error -- The tape drive cannot read a block of data from the tape. This could exist if: (1) The tape drive head is dirty. ++ Clean the tape drive head as described in your Tape Transport Technical manual. (2) The tape is damaged, folded, split or wrinkled. ++ Try a new tape. (3) The cabling between the tape drive and the interface is installed incorrectly or faulty. ++ Reseat the cable and try again, making sure no pins are bent in the 62 pin connector. If you have tried everything please contact a qualified technician to help correct the problem.
5	Correctable Tape Error Encountered -- This is caused by a single track dropout on a write. Clean your tape drive head, and tape path as described in the Tape Transport Technical Manual.
6	Reverse Operation Attempted at B-O-T -- You cannot tell the Drive to skip backwards while at B-O-T. -- To unload the tape type: **TPOS TAPE:/UNL**
7	Internal Software Error - Invalid Call -- You are possibly using a contaminated copy of the software utility. Make a new copy from your software distribution disk and try again.
8	Interface Error - Zero Length Record -- Clean the Tape head as described in your Tape Transport Technical Manual, and try running a TBACKUP with a scratch tape. -- More often than not, this error occurs due to trying to read a tape density that is not compatible with you tape drive. ++ If this tape comes from another party, please see if they can rewrite the tape at a density your drive can read. -- This problem can also occur due to bad Blocks on the tape. ++ Try skipping forward a few blocks and reattempt to read.
9	Incomplete DMA - Block Length Overflow -- Attempting to read a block size greater then 64k bytes with any utility other than SEISMIC.
11	Interface Error - Lost Command -- Check data cable connections both on the computer side, and on the Tape Drive side, making sure that no pins are bent in the 62 pin connector on the computer side of the cable. -- Drive was taken Offline during an operation.

Code	Explanation
12	Unexpected End of Tape Encountered -- YOU ARE AT THE END OF THE DATA ON TAPE. -- Logical EOT was encountered while performing a forward operation other than /EOT. Example: **TPOS TAPE:/SF:3** When there are less then 3 Files to skip before an EOT mark is encountered.
13	B-O-T Encountered on Reverse Operation -- You CANNOT use /RSB:*n*, /RSR:*n*, or a /RSF:*n* parameters at B-O-T.
14	Noise Block Encountered -- This could be caused by a noisy connection between the tape drive and the computer. -- Some of the CLONES use a chip set with a pseudo DMA controller and may not be fully compatible with a true IBM. ++ Turn off DMA using the TINSTALL program, and retry your application.
15	Tape Write Attempted Without Write Ring -- Make sure you have a Write Ring on the tape reel to which you are trying to write. -- There may be an I/O address conflict with another device. -- If you have a Write Ring on the tape, there may be trouble with the Write Ring Sensor. The WRT ENBL light on the drive should be on. -- **PCT-9L USERS:** Make sure your Write Protect button (FPT) has been pressed and that the light is OFF.
16	Device Not Present -- This error indicates that **THE INTERFACE CARD** was not located at the address given through TINSTALL. ++ Check that the address settings in TINSTALL match the address jumpers on the Controller Card. (See Appendix B) -- The Controller Card may not be installed in the computer. -- There could be a conflict between the Interface Card and another card in the system. -- **ATC-16 USERS:** Make sure the interface card is in a 16 Bit slot and not an 8 Bit slot.
29	Non-Numeric Value on Command Line -- An attempt was made to use an ALPHA character where a NUMERIC value was needed. Please refer to the section which describes operation of the program you are trying to use.
30	Value Out of Range on Command Line -- An attempt was made to use a value other then what is possible. Please refer to the section which describes operation of the utility you are using.

11.2: (...continued)

Code	Explanation
31	Internal Software Error - Type Unknown -- Please contact your Dealer or Catamount.
99	Unable to Access TCONFIG.SYS -- **TCONFIG.SYS** was not found in the system PATH. Please check the PATH command, and place **TCONFIG.SYS** within the one of the subdirectories specified by PATH, or include the Catamount Subdirectory into the PATH command.

11.3: DT Error Codes

Error codes that may be returned by the DT program and their possible causes are as follows:

Code	Explanation
32	Insufficient Memory for DT Program -- Your computers RAM is full. Check to see if you can remove one or more of your TSR (Terminate / Stay Resident) programs. This can be done by changing your AUTOEXEC.BAT file and REBOOTING. -- If you have less than 640k of RAM, you might consider adding more memory to your system.
33	Tape Drive Address Out of Range (0-7) -- You must specify TAPE:, TAPE0: thru TAPE7:.
34	Line Feed Parameter Only Valid After /CR -- Your command line should read something like this: **DT** *filename.ext* **TAPE:/CR/LF** -- Also see the /CR command description in section 5 of this manual.
35	Logical Record Length Too Large -- The Record Length must be between 1 and 65535.
36	Improper File Specification -- The Filename you specified does not conform to standard DOS rules for Filenames, please check your typing and retry
37	/B Parameter Not Allowed on Tape to Disk Transfer -- /B command does not need to be specified to read a tape.
38	(Command Syntax Error) -- Check your syntax in the manual or with the displayed Help screen and try the command again.

11.3: (...continued)

Code	Explanation
39	EOT Skip Not Allowed on Tape to Disk Transfer -- /EOT is used to append to the end of the tape with DT Disk to Tape transfers. It is correct to type: **DT** *filename.ext* **TAPE0:/EOT** But not: **DT TAPE0:/EOT** *filename.ext*
40	Unable to Open Output File For Append -- The File Specified was not found. -- Check the Directory and PATH. -- If the File does not exist you can not use the append mode.
41	Record Length of Zero Not Allowed -- The Record Length must be between 1 and 65535.
42	Carriage Return Processing Not Allowed with /U Mode -- This error occurs on TAPE to DISK transfers. -- Either select /U or /CR.
43	Record Length Not Allowed with /U Mode -- This error occurs on TAPE to DISK transfers. -- Either select /U or /R:*n*.
44	Tab Expansion Not Allowed Without /CR Mode -- This error occurs on TAPE to DISK transfers. -- You must specify /CR to use /T
45	Bad or Undecipherable Path or File Name -- The File specified was not found. -- Check the PATH and Directory for file. -- On Tape to Disk transfers, the File may exist, but as read only.
46	DOS Error - Too Many Files -- Your Disk has too many file entrys. -- Try a different disk or subdirectory.
47	File Already Exists as Read Only -- The File specified was found as READ ONLY. -- Either use a different filename or subdirectory.
48	Tab Expansion Not Allowed Without /CR Mode -- This error occurs on DISK to TAPE transfers. -- You must specify /CR to be able to use /T.

Code	Explanation
49	/EOT AND /SF:n Conflict - Only One May Be Specified -- Either select /EOT or /SF:*n*.
50	Logical Record Length Not Allowed with /U Mode -- This error occurs on DISK to TAPE transfers. -- Either select /U or /R:*n*.
51	/CR and/or /LF Not Allowed with /U Mode -- This error occurs on DISK to TAPE transfers. -- Either select /U or /CR/LF.
52	Block Size Not a Multiple of Record Length -- Make sure the Block Length /B:*n* is a multiple of the Record Length /R:*n* or use /U.
53	DOS Cannot Access File -- This normaly occurs when you try to access a Drive that does not exist or File that has limited access, such could be the case with network system files.
54	Disk File Not Found -- The Source File was not found in the PATH. -- Check your Directory and change your PATH if needed.
55	Logical Record Length Incompatible With Tape Block Size -- An attempt was made to read from the tape with /R:*n* other than a multiple of the Block size on tape. -- You may need to use /U or /SHORT if you are dealing with a tape of variable block sizes.
56	Disk Write Error - Check Disk and Free Space -- The disk is either full or faulty.
57	Disk File Size Not Multiple of Record Size -- The Disk File must be a multiple of the record size. -- Try using /U or use /CR/LF.
58	Error Reading Disk File - Transfer Aborted -- Check your disk with the DOS command **CHKDSK /F**.
59	Tape Drive Went Not Ready During Command -- see error code 11 in section 11.2 (common error codes).
60	Program Aborted by Operator -- The operator used CTRL-BREAK to stop the operation of the selected utility.

11.4: TBACKUP Error Codes

Error codes that may be returned by the TBACKUP program and their possible causes are as follows:

Code	Explanation
32	Insufficient Memory Available for TBACKUP -- Your computer's RAM is full. Check and see if you can remove one or more of your TSR (Terminate / Stay Resident) programs. This can be done by changing your AUTOEXEC.BAT file and REBOOTING. -- If you have less then 640k of RAM you might consider adding more memory to your computer.
33	Program Aborted by Operator -- The operator used CTRL-BREAK to stop the operation of the selected utility.
34	Tape Drive Address Out Of Range (0-7) -- You must specify TAPE:, TAPE0: thru TAPE7:.
35	Decimal Value Out of Range in Command Line -- Check the Value you specified for /C:*n* and try again.
36	Invalid Date Specification -- Date must be specified as below: **TBACKUP** *filename.ext* **TAPE:/D:***mm-dd-yy* Where *mm* = Month (1 - 12) *dd* = Day (1 - 31) *yy* = Year (80 - 99) Example: **TBACKUP *.* TAPE0:/D:03-05-88** Only files Dated March 5th 1988 and latter will be written to tape.
37	Invalid File Specification for /EX: -- /EX: must be specified as below: **TBACKUP** *filename.ext* **TAPE:/EX:***filename.ext* Where *filename.ext* = Any DOS filespec.
38	(Command Syntax Error) -- Check your typing and try again.
39	/DISKS: Not Valid With d: -- Do not specify source device d: in the command line when you use the /DISKS Paramenter.
40	Disk Drive Designator (d:) Out of Range -- (d:) must be an alpha character from A to the last drive on your system or network.

Code	Explanation
41	Invalid Drive Specification -- The Drive specification must be a valid drive between A: and the last drive on your system or network. Example: **TBACKUP C:*.* TAPE:**
42	Invalid Path Specification -- The Path specified must exist on the drive specified. Example: **TBACKUP C:\DOS*.* TAPE:**
43	Tape Label Not Valid - Append Not Allowed -- The Tape you are trying to Append has not been used as a BACKUP Tape before, and needs to be initialized. /A cannot be used.
44	Error in Test Data -- The Tape Subsystem has failed its initial test because of a device conflict or hardware failure. ++ Consult Installation Procedure (Chapter 2). ++ See code 46. ++ Consult Catamount.
45	Length Error in Test Record -- The Tape Subsystem has failed its initial test because of a device conflict or hardware failure. ++ Consult Installation Procedure (Chapter 2). ++ See code 46. ++ Consult Catamount.
46	File Mark Error on Test -- The tape drive head could be dirty. ++ Try cleaning the head as described in your Tape Transport Technical Manual. -- The tape may be damaged. ++ Try using a new tape. -- The tape may not be useable at the density at which you are attempting to use it. ++ Try a different tape. -- The data cable might not be attached correctly. ++ Try reseating the cable both at the computer and at the Tape Drive, and check to make sure that no pins have been bent in the 62 pin connector at the computer.
47	Tape Ending Label Not Valid For Append -- The tape to which you are trying to Backup to, does not have the proper Ending Label to be able to Append. Either the tape has not been used for a backup, or it was written with a different density.

11.4: (...continued)

Code	Explanation
48	Internal Directory or Path Error -- Use the DOS command **CHKDSK /F** to check your disk drive.
49	Internal Disk Directory Error - Unable to Close File -- A file is held open by another user on the computer system, or the computer has a legitimate problem requiring profesional service. ++ Use the DOS command **CHKDSK /F** to check for problems. ++ On a network, try to make your backup at a time when others do not need to use the system.
50	Internal Disk Error -- Use the DOS command **CHKDSK /F** to check your disk drive.

11.5: TRESTORE Error Codes

Error codes that may be returned by the TRESTORE program and their possible causes are as follows:

Code	Explanation
32	Insufficient Memory Available for TRESTORE -- Your computer's RAM is full. Check and see if you can remove one or more of your TSR (Terminate / Stay Resident) programs. This can be done by changing your AUTOEXEC.BAT file and REBOOTING. -- If you have less then 640k of RAM you might consider adding more memory to your machine.
33	Program Aborted by Operator -- The operator used CTRL-BREAK to stop the operation of the selected utility.
34	(Command Syntax Error) -- Check your typing and try again. -- If you cannot find any mistakes, check in the Chapter which discusses the utility you are using.
35	Invalid Drive Specification -- The Drive specification must be a valid drive between A: and the last drive on your system or network. Example: **TRESTORE TAPE: C:*.***
36	Invalid Path/File Specification -- The Path/File specified must exist on the drive specified. Example: **TRESTORE TAPE: C:\DOS*.***

11.5: (...continued)

Code	Explination
37	Internal Path Specification Error
38	/DISKS: not valid with /DISK: -- Either select /DISK: or /DISKS:.
39	Bad Filename Specification -- The filename used was not a valid filename. ++ Check your typing and make sure the file exists.
40	Tape Label Record Invalid -- The tape was not created with TBACKUP. -- The Label at the beginning of the tape has been damaged or written over.
41	Path Length Too Long -- The Path length must be less then 64 bytes.
42	Internal DOS Error - Check Directory -- Use the DOS command **CHKDSK /F** to check your disk drive.
43	No Matching Files Found -- No files were found on the tape that match the filename and the path you specified. ++ Use **TDIR TAPE:** to see what is on the tape. -- You may have typed the /PATH: or /DISK: parameters wrong.
44	Disk Error on Write - Check Disk and Free Space -- Either the disk is full or there is no room left for directory entrys or the disk is faulty.

11.6: TDUMP Error Codes

Error codes that may be returned by the TDUMP program and their possible causes are as follows:

Code	Explanation
32	Insufficient Memory Available for TDUMP -- Your computers RAM is full. Check and see if you can remove one or more of your TSR (Terminate / Stay Resident) programs. This can be done by changing your AUTOEXEC.BAT file and REBOOTING. -- If you have less then 640k of RAM you might consider adding more memory to your machine.
33	Program Aborted by Operator -- The operator used CTRL-BREAK to stop the operation of the selected utility.

Code	Explanation
34	Record Length of Zero Not Valid -- A Record Length greater than Zero must be specified.
35	Skip Count of Zero Not Valid -- This error occurs with the SKIP BLOCK function. -- A value greater than Zero must be specified as the SKIP count.
36	Skip Count of Zero Not Valid -- This error occurs with the SKIP FILE function. -- A value greater than Zero must be specified as the SKIP count.
37	/SB:*n*, /RSB:*n* and /RSF:*n* Not Valid with /ANSI -- To select the /SB:*n*, /RSB:*n* or /RSF:*n* you cannot select /ANSI, as /ANSI selects its values from the header found on tape.
38	/LENGTH: Value Not in Range -- The value must be between 0 and 255 inclusive.
39	(Command Syntax Error) -- Check your typing and try again. If the error persists, check the chapter which describes operation of the utility you are using.
40	Bad or Undecipherable Output Device Path or File Name
41	DOS Error on Output Device - Too Many Files -- You may need to respecify the **FILES=***n* command in the CONFIG.SYS file. Add 2 or 3 to the current value and reboot.
42	Output File Already Exists as Read Only -- You cannot write to a Read Only File. ++ Either use a different file name, or a different subdirectory.
43	ANSI VOL1 Label Record Missing -- The tape may not have been written using ANSI format and the ANSI Lable is not available.
44	Line Length Too Long
45	Error in Label Record Length -- The tape may not be ANSI format or the ANSI Label was corrupted.

11.7: TDIR Error Codes

Error codes that may be returned by the TDIR program and their possible causes are as follows:

Code	Explanation
32	Insufficient Memory Available for TDIR -- Your computers RAM is full. Check and see if you can remove one or more of your TSR (Terminate / Stay Resident) programs. This can be done by changing your AUTOEXEC.BAT file and REBOOTING. -- If you have less then 640k of RAM you might consider adding more memory to your machine.
33	Program Aborted by Operator -- The operator used CTRL-BREAK to stop the operation of the selected utility.
34	(Command Syntax Error) -- Check your typing and try again. If the error persists, check the chapter which describes operation of the utility you are using.
35	Disk/Path/File Specification Not Valid
36	Bad or Undecipherable Output Device Path or File Name
37	DOS Error on Output Device - Too Many Files -- You may need to respecify the **FILES=*n*** command in the CONFIG.SYS file.
38	Output File Already Exists as Read Only -- You cannot write to a Read Only File. ++ Either use a different file name, or a different subdirectory.
39	Tape Label Record Not Valid -- The tape was not created with TBACKUP. -- TBACKUP label at the beginning of the tape has been damaged or written over.
40	No Matching Files Found -- No Files were found on the tape directory which matched the search filename specified.

11.8: TPOS Error Codes

Error codes that may be returned by the TPOS program and their possible causes are as follows:

Code	Explanation
32	Skip Count of Zero Not Valid -- You must specify a value from 1 to 65535.

11.8: (...continued)

Code	Explanation
33	(Command Syntax Error) -- Check your typing and try again. If the error persists, check the chapter which describes operation of the utility you are using.

11.9: IOCTL Error Codes

Error codes that may be returned by the IOCTL program and their possible causes are as follows:

Code	Explanation
32	No Such Device
33	Error in Device Status Routine
34	Device Does Not Exist as Character Device
35	Device Not Enabled for IOCTL -- Make sure that you are specifying a valid Device. Example: **IOCTL TAPE0:/BOT** **IOCTL G:/BOT** -- Where G: is the first tape drive being used in disk emulation mode.
36	Error on IOCTL Transmit
37	Error on IOCTL Receive
38	Error in Listing Result
39	Error on Listing Result -- Either **TDS.DEV** or **TDS.COM** is not installed correctly. Check your CONFIG.SYS for a line that reads: **DEVICE = TDS.DEV** -- Remove other device drivers from the system which may conflict.
40	(Command Syntax Error) -- Check your typing and try again. If the error persists, check the chapter which describes operation of the utility you are using.

11.10: BPS Installation Error Codes

Error codes that may be returned by the BPS installation program and their possible causes are as follows:

Code	Explanation
32	New BPS Image Not Installed -- Either the old BPS is still installed, or no BPS was installed.
33	Block Size of Zero Not Allowed -- Use a value from 1 to 65535.
34	Invalid Maximum Drive Address (Must Be 0-7) -- Keep the value from TAPE:, TAPE0: to TAPE7:.
35	Maximum Block Size Must Not be Zero -- Use a value from 1 to 65535.
36	Insufficient Memory to Allocate Buffers -- Your computer's RAM is full. Check and see if you can remove one or more of your TSR (Terminate / Stay Resident) programs. This can be done by changing your AUTOEXEC.BAT file and REBOOTING. -- If you have less then 640k of RAM you might consider adding more memory to your machine.
37	(Command Syntax Error) -- Check your typing and try again. If the error persists, check the chapter which describes operation of the utility you are using.

11.11: TDS Installation Error Codes

Error codes that may be returned by the TDS installation program and their possible causes are as follows:

Code	Explanation
32	New TDS Image Not Installed -- Either the old TDS is still installed, or no TDS was installed.
33	Invalid Maximum Drive Address (Must Be 0-7) -- Keep the value from TAPE:, TAPE0: to TAPE7:.
34	Maximum Block Size Must Not be Zero -- Use a value from 1 to 65535.

Code	Explanation
35	Insufficient Memory to Allocate Buffers -- Your computers RAM is full. Check and see if you can remove one or more of your TSR (Terminate / Stay Resident) programs. This can be done by changing your AUTOEXEC.BAT file and REBOOTING. -- If you have less then 640k of RAM you might consider adding more memory to your machine.
36	(Command Syntax Error) -- Check your typing and try again. If the error persists, check the chapter which describes operation of the utility you are using.

Appendix A

Tape Primer

Introduction

For more than thirty years magnetic tape has provided a standard reliable interchange medium for data processing. While its acceptance is almost universal due to its longevity, some of the protocols and standards have become obscure and almost archaic in todays world of data processing.

For those who must work effectively with 1/2" tape, this primer is provided to discuss in detail the technology, standards, protocols and methods used in working with 1/2" tape.

In many cases, the reader will hold an interest in only one or two of the areas discussed and should feel free to scan over detailed technical information covering matter not relevant to his application.

Appendix A

The Origins of 1/2" Tape

To fully cover the subject of 1/2" tape, we go back to its predecessor in computer applications, the Hollerith card. Adapted by Herman Hollerith from punched cards such as the cards used to control the Jacquard Loom (and other manufacturing equipment), the Hollerith card was used to store and process numeric information, establishing its claim to fame through its use in the 1920 United States Census.

"Tab" (tabulation) equipment developed to process the Hollerith card was well established prior to the appearance of the conventional, Von Neumann computer. Unit record equipment, as it was called, mechanically, electromechanically and (eventually) electronically processed the information recorded on the Hollerith card. Tab machines were designed to accumulate totals, sort, count, reformat, punch, list and otherwise process data appearing on Hollerith cards (tab cards). As the concept of the Von Neumann machine invaded the domain of the tab equipment, the need for random and sequential reuseable storage media presented itself, and was rapidly satisfied with existing magnetic recording technology adapted primarily from audio applications. Magnetic disks, drums and tapes in all imaginable configurations soon appeared on the computer scene. 1/2" tape established an early lead in information storage, and has held this lead to this day for the purpose of storing and processing large amounts of information.

As it originally appeared, 1/2" (and other widths) of tape were used to store information in a manner that differed considerably from the methods used today. Many early machines were decimal machines, recording information using combinations of the numbers 1, 2, 3 and 6 generating the decimal digits 1 through 9. Later decimal machines used binary coded decimal, adopting bits (binary digits, i.e., 1, 2, 4 and 8) to store decimal information. Data densities were usually much lower, and the concept of bytes had not yet appeared.

Through evolution, tapes have progressed in density from 200 bits per inch (bpi) through 556, 800, 1600 up to 6250bpi. Tapes which once held four tracks of data expanded first to to seven and then to nine. Newer technologies, such as the recently announced IBM 3480 cartridge tape drive, use densities several times higher and as many as 18 tracks. Binary machines have all but entirely replaced decimal machines.

Basic Tape Technology

Most digital magnetic storage technologies work on the principal of storing information as patterns of magnetization embedded in a magnetic coating applied to a substrate. For most 1/2" tape formats, sections of this magnetic coating are magnetized to saturation in one of two standard, opposite directions, by convention selected such that tape is considered to be magnetized in the erase direction if it is magnetized such that the outer rim edge of the tape is a north seeking pole.

The relationship of electrical properties to magnetic properties is such that the movement of electrical charge creates magnetic fields, while changes in magnetic fields induce electrical fields in conductors that cause movement of electrical charge. Thus, the tape coating is magnetized in one direction or the other by switching current through the head coil in one direction or the other, and information is retrieved from tape by moving the tape past the head coil, detecting current bursts induced by changes in the magnetic field. Note that these current bursts are only produced by changes in the magnetic field (flux changes). If the tape is magnetized uniformly in any direction, the circuitry is unable to detect the magnetization.

In virtually all 1/2" tape recording processed on computers today, seven or nine tracks of data are recorded in parallel along the tape, with a sepearate head coil used for each track. Many tape drives use separate read, write and/or one or more erase head coils so that data may be read and validated after it is written without backspacing the tape, and in some cases, erroneous data erased after it is written. A head containing separate read and write coils is referred to as a dual gap head. The capability to erase with a separate head after the read is referred to as downstream erase.

Formats

To record information onto tape in a standard, interchangable manner, the tape is formatted in accordance with one of several formatting standards. The NRZI (Non-Return to Zero Invert) format is commonly used at data densities of 200, 556 and 800 bpi on seven and nine track tapes. The PE (Phase Encoded) format is a nine track format normally used at 1600 bpi, with some newer tape drives utilizing PE at 3200 bpi. GCR (Group Coded Records) encoding is also a nine track format used at a data density of 6250 bpi. A basic description of each of these formats follows. Additional formatting features of each of these recording formats will be discussed later.

NRZI records a logical 1 as a flux reversal (a reversal of magnetic field, or magnetization direction) from magnetization in whichever direction the tape was magnetized prior to the reversal to magnetization in the opposite direction. A 0 is noted as the absence of such a reversal.

PE data is recorded by beginning a logical 1 with a flux change from the non-erased direction to the erased direction, and recording a logical 0 as a flux change from the erased direction to the non-erased direction. To facilitate this, a second flux reversal occurs for each recorded bit when the following bit is of the same value, in essence resetting the magnetization so that it may be again reversed.

GCR records data in the same manner as NRZI, however, data is manipulated by the formatter such that data is first divided into subgroups of four bits which are then mapped into five intermediate bits for the purpose of recording. (When read back, these five bits are then mapped back to the original four). This mapping is used to eliminate certain bit patterns, notably patterns yielding more than two zero bits in a row and patterns with more than one zero bit beginning a subgroup or ending a subgroup, to eliminate the possibility of more than two zero bits in a row between two consecutive subgroups. Bits are mapped as follows:

Data Bits	Recorded Bits
0000	11001
0001	11011
0010	10010
0011	10011
0100	11101
0101	10101
0110	10110
0111	10111
1000	11010
1001	01001
1010	01010
1011	01011
1100	11110
1101	01101
1110	01110
1111	01111

In addition to this mapping of subgroups of four data bits into five bits, subgroups of seven data bits along each track are expanded to eight data bits by computing eight inclusive polynomials in parity for all 64 bits (eight bits on eight tracks) yielding eight correction code bits which are distributed and recorded among the eight tracks.

In all, GCR encoding records groups of seven data bits as ten bit cells. Additional reductions in the effective data density are the result of resynchronization bursts recorded for every 158 subgroups of seven bits. The purpose of these resynchronization bursts is discussed later.

Each of these formats has its own merits and weaknesses. Because the geometry and sensitivity of the recording heads and the sensitivity of the magnetic tape media limit how densely information can be stored, formats which record the most data with the lowest corresponding number of flux changes yield the highest data capacities for a given flux reversal density. For the NRZI format, where at most one flux reversal is required to store one bit of data, the data density is equal to the maximum flux reversal density. Thus, tape heads, tape media and read/write circuitry must only be able to handle flux reversal densities of 800 flux changes per inch (fci) to record 800 data bits per inch (bpi).

With the PE format, two flux reversals may be required to store each data bit, thus requiring heads, media and circuitry capable of flux reversal densities of 3200 fci to support data recording at 1600 bpi.

With the GCR format, as with the NRZI format, a recorded 1 requires one flux reversal, and a 0 does not require any. With GCR however, due to the expansion of four data bits to five, and yet an additional correction bit generated for each seven data bits, together with other format complexities, when recording is performed at a density of 9042 fci (as is standard), only 6250 data bits are recorded per inch. Note that despite this, nearly four times the GCR data is recorded per inch of tape as with PE, while the flux reversal density has not increased threefold.

Parity

Common to all three formats discussed here, one track of tape data is reserved to contain what is called a "parity" bit. Logicians describe parity as a polynomial, however, it is a polynomial generated by logically "summing" (using the exclusive or "XOR" function) the data bits across the tape tracks, as opposed to mathematically summing them. Assuming for the purpose of this discussion that nine tracks of data are being used, a byte of data is recorded as eight bits, with one bit being recorded in each of the eight data tracks across the tape. The ninth track contains the parity bit, computed to be the complement of the sum of the eight data bits. Simply put, for "odd parity" the parity bit is set to 1 if an even number of data bits are 1s, 0 if an odd number of data bits are 1s.

Generating parity in this fasion guarantees that the logical sum of all bits across the tape (including the parity bit) will be odd, providing a method of determining if an error has occured on any single bit. If any single bit were to be misread, the parity checking circuitry of the tape formatter would detect an even sum and immediatly know that an error had occured.

In addition to guaranteeing that a single bit error has not occured, odd parity serves an additional function for NRZI tape. Because a 1 parity bit results when a byte of all zero bits occurs, at least one 1 bit, therefore one flux transition, occurs for each byte written. This eliminates the possibility of having a string of bytes containing all zeroes recording a stretch of continuously magnetized tape, which by virtue of the fact that the tape would contain no flux transitions, would give no accurate indication of how many zero bytes have gone by. This advantage turns into a disadvantage when error correction (discussed later) is required, however. Even parity guarantees that at least two 1 bits occur on each character (except where characters containing all zeros occur), providing a 1 bit even if a track has dropped out. Bytes containing all 0s are not usually allowed on NRZI tape.

Alignment

This concern is unique to NRZI formats. NRZI recording is alone in using flux transitions occuring along one track to decode data (0 bits) occuring on other tracks. All bits within a byte of NRZI data must occur at very nearly the same time, requiring precision mechanical alignment of the tape head and tape path. Because at least one flux transition occurs for each data bit in the PE format, bits are broken out independently along each track before being assembled into bytes across the tracks. This allows for one (or more) tracks to be ahead or behind other tracks on the tape, with the only restriction on how far ahead the first bit may be of the last bit in the same byte being how many bits the particular formatter is designed to buffer. Many PE tape formatters allow several times the maximum standard data skew (deviation from alignment) without error.

Error Correction

In addition to the ease of "deskewing" (separating data along the tracks) data on PE tapes, the recording of at least one flux transition for each data bit also provides an additional method of error detection. If no flux change occurs within the expected period of time for a PE data bit, it can be assumed that the data from the track in which the flux transition is missing is not available. This is referred to as a track "drop out". If a single track of PE data drops out, the content of the missing track can be reconstructed by logically summing all remaining tracks, including the parity track. If the resulting sum is a 0, it can be assumed that the data bit for the missing track is a 1, while if the sum yields a 1 the missing bit must have been a 0.

As with the PE format, GCR encoding also deskews data on a track by track basis. Because the creation of the five bit subgroups guarantees that no more than two 0 bits occur in a row, a flux transition will occur at least every three character periods. By using flux transitions to "realign" the timing of data, it is possible to use the time between flux transitions to determine if one or two 0 bits have passed. If too long a period passes without a flux transition, a drop out is assumed to have occured.

As the length of a block of data increases, the likelihood of an error also increases. Correspondingly, so does the likelihood of multiple errors. The PE format can only correct a single track with parity, and if a second track drops out, data becomes unrecoverable. The GCR format compensates for this by embedding a resynchronization burst every 1106 data bytes (158 7-bit subgroups) if more than 1112 bytes are to be recorded between synchronization bursts. A resynchonization burst consists of a pattern readily detected by the formatter logic which allows a track to resynchronize itself with the other tracks, allowing error correction to restart itself within a block. This eliminates the error correction's sensitivity to long blocks of data as occurs with the PE (and NRZI) format.

The GCR format provides three independent levels of error detection and/or correction. First, dropouts can be detected by the presence of data patterns that deviate from the sixteen five-bit patterns recorded. Next, one byte of error correction data is stored for every 7 bytes of tape data. This byte is generated using an inclusive error correction polynomial, and may be used to correct any bad bit occuring withing the eight byte subgroup. (The correction byte itself participates in the correction logic, so that errors in the correction byte are also detected). Finally, parity across all tracks is stored.

The NRZI format stores two error detection characters at the end of each tape block. These characters are generated to complement two independent error detection polynomials, and are referred to as the CRC (Cyclic Redundancy Check) character and the LRC (Longitudinal Redundancy Check) character. When a tape error is detected, the LRC character may be used to determine the track containing the error and the corrected data verified by recomputing the CRC polynomial, which will yield a zero result for correct data.

General Tape Organization

Beyond the methods of recording actual data bits onto tape, certain additional protocols are used to identify the type of format used on a tape, identify and separate data records and data files, indicate the beginning of the physical tape, the end of the physical tape, and the end of data recorded on the tape.

A standard reel of tape begins with at least 14 feet of unused leader. (This leader may be trimmed down at various times as it becomes damaged. If it becomes too short, data should be copied to a new tape and the leader stripped off. A new reflective marker should then be applied about 25 feet from the cut end of the tape). A reflective marker is applied to the side of the tape away from the hub, indicating the end of the leader and the beginning of the recording surface of the tape. (Reflective markers are placed on the "back side" of the tape, not the side containing the "oxide" used to record data).

When the first record of a PE tape is written, an identification burst is recorded as a series of 1600 fci flux transitions along track 4. A similar burst recorded at 3200 fci along track 6 is used to indicate that a GCR tape is being written. In both of these cases, all other tracks are held in the erase direction. GCR tapes follow this identification burst with an additional identification pattern involving various flux reversals on all of the tracks.

Following the identification bursts, if any, is an "interblock gap". These gaps are created by holding erase current across all heads. Normally, interblock gaps are used to separate data records. Interblock gaps must be at least .5" in length on NRZI and PE tapes and .3" on GCR tapes. Interblock gaps may be longer than the minimum but must not exceed twelve feet for the tape to conform with standards.

The beginning of a data block is marked by a "preamble". NRZI, PE and GCR preambles differ, but in general, contain a series of oscillations (flux reversals) to initialize the read logic, a few maximum frequency oscillations, and a special pattern indicating that the actual data begins immediately following. These patterns synchronize the deskewing logic so that all tracks are able to separately start their data decoding in phase with each other. A reverse of the preamble pattern (called a "postamble") is recorded at the end of the data block, so that the tape may be read backwards in the same fashion as it is read forward.

Data is recorded between the preamble and the postamble, with resynchronization bursts recorded periodically in GCR records.

After the postamble, an interblock gap, as described above, appears. Another data block may then follow, complete with its preamble and postamble.

In addition to separating data blocks, interblock gaps also allow for data correction. When errors occur while writing tape, (usually caused by bad sections of tape which have been creased or had the oxide scraped off), the section of tape which does not correctly store the recorded information is erased, becomming a part of an extended interblock gap. The data is then rewritten further down the tape.

Interblock gaps also provide a method of updating, or changing, data on an existing tape. Tape blocks in the middle of a tape may be rewritten by starting erase current to the head in the middle of an interblock gap, writing a record as normal, and turning off the erase current in the middle of the following interblock gap. Also, entire blocks may be erased from a tape by holding erase current from the middle of the preceeding interblock gap until the middle of the following interblock gap, effectivly blending the record into an extended interblock gap. Note that when rewriting a block, normal write error correction must not take place, as there is no space to extend an erase gap over a bad section of tape.

Data blocks are separated into files (and sometimes label records, as described later) by writing "tape marks" (or "file marks") to the tape. Tape marks are recorded by writing an erase gap of roughly 3 1/2" followed by a few hundred high frequency flux reversals (recorded at the maximum frequency used by the format) along tracks 1, 2, 4, 5, 7 and 8, while holding tracks 3, 6 and 9 in the erased direction.

By convention, the logical end of the tape (the point beyond which no further useful data is recorded) is marked by recording two consecutive tape marks.

The physical end of tape is indicated by a reflective marker placed on the hub side of the tape. This is used to instruct the formatter (and eventually, the software) that the end of the tape is near and that recording must cease. Tapes must provide at least four feet of useable tape after the EOT mark.

Storing Data Files

Now that the physical aspects of storing data on 1/2" tape have been described, it's time to discuss how to make use of the data in computer and data processing applications.

Once again looking back through the history of data processing, 1/2" tape followed the Hollerith card as a storage medium. Initially, 1/2" tape was used for unit record storage in a fashion very similar to Hollerith cards. Even today, it is most likely that a tape contains blocks of 80 byte records, representing card images, or 121 or 133 byte records containing images of printer data, complete with printer control characters.

By tradition, tape data is usually stored as records, with each record containing a complete set of data for a particular transaction, item or event. Because tape record sizes are not so limited as those of punched cards, the relationship between the original data and how it is stored on tape is stronger than with punched cards.

There are very few penalties in adapting the tape record size toa size most appropriate for an application. Because of this, there are no standard lengths for tape records, except for applications where card images or printer data is to be stored. This is unlike disks, which are frequently organized with a fixed number of sectors of a fixed size. (Older, large system disk drives use variable length records in a fashion very similar to tape).

Blocking

Because interblock gaps require space on the tape (1/3" to 1/2"), tape records are frequently combined into blocks containing more than one record. This is referred to as "blocking" records. Logical records are combined into physical blocks for recording on tape, and then separated into logical records after being read back. This combination and separation is normally performed by the computers operating system or I/O software.

An example of how blocking helps to increase the capacity of a tape would be where 80 byte records are to be recorded. Assuming that a PE tape is to be written (at 1600 bpi), 80 bytes would consume 80/1600ths of an inch of tape, or 1/20". But when you add to this the space occupied by the 1/2" interblock gap (and a little bit more for the preamble and postamble), a foot of tape would only be able to contain about 22 records. If instead, these records were combined into, say, blocks of 40 records (3200 bytes each), nearly 200 records would fit on a foot of tape, yielding an eightfold increase in storage capacity.

This format is usually called "fixed blocking", with a fixed number of fixed length records recorded as a single block.

Some operating systems support variable length records. Unformatted variable length records are sometimes used where the length of the record is equal to the length of the block. This record format is common for storing disk sector images, particularly for binary program modules normally residing on variable length disk sectors.

Other standards exist for variable length records which utilize a record prefix containing a count of the record length. This prefix may be in binary, as used by the machine, or recorded as a decimal represented in a display character set, such as EBCDIC or ASCII, as discussed later. In some instances these counts are contained in two bytes, others four bytes. IBM uses variable length records with a four byte binary prefix. (A caution is given here to programmers using mini or microcomputers to read these tapes. IBM stores binary values with the most significant byte first. Most mini and microcomputers store binary values with the least significant byte first.)

When variable length records are blocked, they are usually blocked with an additional prefix preceeding the record length prefix. This prefix contains the block length.

Additional formats begin to look more like disk formats. Variable length and fixed length spanned records allow data records to span (appear partly on one or more) blocks. The physical block length when using these formats becomes entirely independent of the logical record length. These formats are usually not suitable for reading backwards.

Character Data Representation

When data was first recorded on 1/2" tape by computers, only decimal data was stored and processed. This limited the data representations to one simple format, but also limited the utility of the data being stored. Data formats changed as computers went to Binary Coded Decimal, storing binary digits directly instead of decimal components. The Binary Coded Decimal format was extended to include, in addition to the 1, 2, 4 and 8 bits, two additional bits, then referred to as the "A" and "B" bits, used to expand the code to include alphabetic characters and symbols. The six bit BCD (Binary Coded Decimal) code corresponded to codes generated on earlier keypunches (such as the IBM model 026) and together with a parity bit, was recorded on seven track tapes.

Various computer manufacturers used similar, if not identical, six bit display codes. Todays remaining prevalance of seven track tape results from the ongoing use of computers that use six bit display character representations.

In the telecommunications industry (primarily wire service, telegraph and teletype) at about this same time a seven bit code, ASCII, the American Standard Code for Information Interchange, was beginning to establish itself over the five-bit Baudot code, which used a transmitted shift character to go back and forth between alphabetic and numeric characters. ASCII provided an excellent representation of character and text data, and began to also establish a hold in the computer market.

IBM apparantly felt the need to enhance the six bit BCD code and provide more symbols, lower case characters, and special codes for communications protocols. The resulting Extended Binary Coded Decimal Interchange Code, EBCDIC, is an eight bit code, which corresponded to a new code set for keypunches, replacing the 026 with the 029.

When processing tapes today, it is common to find ASCII and EBCDIC character representations on nine track tape, and one still occaisonally encounters BCD on seven track tapes.

In all character formats data is encoded on 1/2" tape by writing 1s and 0s in the same fasion binary numbers are recorded. To understand how characters are recorded, first we must understand how binary information is recorded. For the purpose of this discussion, we will assume that nine track tape is being used.

Eight of the nine tracks on the tape are used to record data bits and one track is used to record the parity bit. By convention, we assign the parity bit to the fourth track on the tape. The most significant bit, bit 0 in magnetic tape parlance, is assigned to the seventh physical track. The next most significant bit, bit 1, appears on track 6, bit 2 on track 5, 3 on 3, 4 on 9, 5 on 1, 6 on 8, and the least significant bit, bit 7, is recorded on track 2.

A binary number is interpreted as (bit 7) + (2 x bit 6) + (4 x bit 5) + (8 x bit 4) + (16 x bit 3) + (32 x bit 2) + (64 x bit 1) + (128 x bit 0). This results in a number having a value between 0 (if all bits are 0) and 255 (if all bits are 1), inclusive. Thus, a byte of data assumes a value ranging from 0 through 255.

(Again, a caution to mini and microcomputer programmers. The convention for numbering bits on 1/2" tape differs from that used for numbering bits on most mini and microcomputers. 1/2" tape standards refer to the most significant bit of a byte as bit 0 and the least significant bit as bit 7, as do the specifications of most large computers. On most mini and microcomputers, the bit designation corresponds to the power to which 2 is raised to generate it's corresponding value).

When characters are written to tape, they are represented as bit patterns on the tape in a manner identical to the way binary values are recorded. Characters are distinguished from binary values only by the meaning implied by the programmer. Thus, to determine whether a byte of information is to be interpreted as a binary numeric value or as a character of alphanumeric information, one must know how the originator of the information intended for it to be interpreted.

Appendix A

Character data can be thought of as directly corresponding to binary values. In the case of nine track tape where an eight bit character set, such as EBCDIC, is to be written, the binary byte values of 0 through 255 correspond to the 256 EBCDIC characters.

In each of ASCII, EBCDIC and BCD, different binary values are recorded for each character. An "A" in ASCII is recorded as a 65. To record an "A" in EBCDIC, 193 is used. BCD records a 49 for an "A".

The following pages contain a chart of binary, octal, hexadecimal, ASCII, EBCDIC and BCD codes as they may appear on computer tape. As it is frequently easier to use the binary number system, or even better yet, octal or hexadecimal to represent the values contained in bytes of data, all four bases are also shown.

Where two or three letters are shown in a column a special character is indicated. These are explained later. The designation "N/P" is used to indicate non-printing characters.

As BCD is a six bit code, there are only 64 possible characters, so only the first 64 lines of this table are present. Likewise, 128 characters of the seven bit ASCII are shown.

Binary	Decimal	Hex	Octal	ASCII	EBCDIC	BCD
00000000	0	00	000	NUL	NUL	:
00000001	1	01	001	SOH	SOH	1
00000010	2	02	002	STX	STX	2
00000011	3	03	003	ETX	ETX	3
00000100	4	04	004	EOT	PF	4
00000101	5	05	005	ENQ	HT	5
00000110	6	06	006	ACK	LC	6
00000111	7	07	007	BEL	DEL	7
00001000	8	08	010	BS	GE	8
00001001	9	09	011	HT	RLF	9
00001010	10	0A	012	LF	SMM	0
00001011	11	0B	013	VT	VT	=
00001100	12	0C	014	FF	FF	≠
00001101	13	0D	015	CR	CR	
00001110	14	0E	016	SO	SO	%
00001111	15	0F	017	SI	SI	|
00010000	16	10	020	DLE	DLE	SP
00010001	17	11	021	DC1	DC1	/
00010010	18	12	022	DC2	DC2	S
00010011	19	13	023	DC3	TM	T
00010100	20	14	024	DC4	RES	U
00010101	21	15	025	NAK	NL	V
00010110	22	16	026	SYN	BS	W
00010111	23	17	027	ETB	IL	X
00011000	24	18	030	CAN	CAN	Y
00011001	25	19	031	EM	EM	Z
00011010	26	1A	032	SUB	CC	]
00011011	27	1B	033	ESC	CU1	,
00011100	28	1C	034	FS	IFS	(
00011101	29	1D	035	GS	IGS	
00011110	30	1E	036	RS	IRS	
00011111	31	1F	037	US	IUS	^
00100000	32	20	040	SP	DS	-
00100001	33	21	041	!	SOS	J
00100010	34	22	042	"	FS	K
00100011	35	23	043	#	N/P	L
00100100	36	24	044	$	BYP	M
00100101	37	25	045	%	LF	N
00100110	38	26	046	&	ETB	O
00100111	39	27	047	'	ESC	P
00101000	40	28	050	(	N/P	Q
00101001	41	29	051	)	N/P	R
00101010	42	2A	052	*	SM	
00101011	43	2B	053	+	CU2	$
00101100	44	2C	054	,	N/P	*
00101101	45	2D	055	-	ENQ	↑
00101110	46	2E	056	.	ACK	↓
00101111	47	2F	057	/	BEL	>

Binary	Decimal	Hex	Octal	ASCII	EBCDIC	BCD
00110000	48	30	060	0	N/P	+
00110001	49	31	061	1	N/P	A
00110010	50	32	062	2	SYN	B
00110011	51	33	063	3	N/P	C
00110100	52	34	064	4	PN	D
00110101	53	35	065	5	RS	E
00110110	54	36	066	6	UC	F
00110111	55	37	067	7	EOT	G
00111000	56	38	070	8	N/P	H
00111001	57	39	071	9	N/P	I
00111010	58	3A	072	:	N/P	<
00111011	59	3B	073	;	CU3	)
00111100	60	3C	074	<	DC4	
00111101	61	3D	075	=	NAK	
00111110	62	3E	076	>	N/P	;
00111111	63	3F	077	?	SUB	
01000000	64	40	100	@	SP	
01000001	65	41	101	A	N/P	
01000010	66	42	102	B	N/P	
01000011	67	43	103	C	N/P	
01000100	68	44	104	D	N/P	
01000101	69	45	105	E	N/P	
01000110	70	46	106	F	N/P	
01000111	71	47	107	G	N/P	
01001000	72	48	110	H	N/P	
01001001	73	49	111	I	N/P	
01001010	74	4A	112	J	¢	
01001011	75	4B	113	K	.	
01001100	76	4C	114	L	<	
01001101	77	4D	115	M	(	
01001110	78	4E	116	N	+	
01001111	79	4F	117	O		
01010000	80	50	120	P	&	
01010001	81	51	121	Q	N/P	
01010010	82	52	122	R	N/P	
01010011	83	53	123	S	N/P	
01010100	84	54	124	T	N/P	
01010101	85	55	125	U	N/P	
01010110	86	56	126	V	N/P	
01010111	87	57	127	W	N/P	
01011000	88	58	130	X	N/P	
01011001	89	59	131	Y	N/P	
01011010	90	5A	132	Z	!	
01011011	91	5B	133	[	$	
01011100	92	5C	134	\	*	
01011101	93	5D	135	]	)	
01011110	94	5E	136	^	;	
01011111	95	5F	137	_		

Binary	Decimal	Hex	Octal	ASCII	EBCDIC	BCD
01100000	96	60	140	`	-	
01100001	97	61	141	a	/	
01100010	98	62	142	b	N/P	
01100011	99	63	143	c	N/P	
01100100	100	64	144	d	N/P	
01100101	101	65	145	e	N/P	
01100110	102	66	146	f	N/P	
01100111	103	67	147	g	N/P	
01101000	104	68	150	h	N/P	
01101001	105	69	151	i	N/P	
01101010	106	6A	152	j		
01101011	107	6B	153	k	,	
01101100	108	6C	154	l	%	
01101101	109	6D	155	m	_	
01101110	110	6E	156	n	>	
01101111	111	6F	157	o	?	
01110000	112	70	160	p	N/P	
01110001	113	71	161	q	N/P	
01110010	114	72	162	r	N/P	
01110011	115	73	163	s	N/P	
01110100	116	74	164	t	N/P	
01110101	117	75	165	u	N/P	
01110110	118	76	166	v	N/P	
01110111	119	77	167	w	N/P	
01111000	120	78	170	x	N/P	
01111001	121	79	171	y	`	
01111010	122	7A	172	z	:	
01111011	123	7B	173	{	#	
01111100	124	7C	174	|	@	
01111101	125	7D	175	}	'	
01111110	126	7E	176	~	=	
01111111	127	7F	177	DEL	"	
10000000	128	80	200		N/P	
10000001	129	81	201		a	
10000010	130	82	202		b	
10000011	131	83	203		c	
10000100	132	84	204		d	
10000101	133	85	205		e	
10000110	134	86	206		f	
10000111	135	87	207		g	
10001000	136	88	210		h	
10001001	137	89	211		i	
10001010	138	8A	212		N/P	
10001011	139	8B	213		N/P	
10001100	140	8C	214		N/P	
10001101	141	8D	215		N/P	
10001110	142	8E	216		N/P	
10001111	143	8F	217		N/P	

Binary	Decimal	Hex	Octal	ASCII	EBCDIC	BCD
10010000	144	90	220		N/P	
10010001	145	91	221		j	
10010010	146	92	222		k	
10010011	147	93	223		l	
10010100	148	94	224		m	
10010101	149	95	225		n	
10010110	150	96	226		o	
10010111	151	97	227		p	
10011000	152	98	230		q	
10011001	153	99	231		r	
10011010	154	9A	232		N/P	
10011011	155	9B	233		N/P	
10011100	156	9C	234		N/P	
10011101	157	9D	235		N/P	
10011110	158	9E	236		N/P	
10011111	159	9F	237		N/P	
10100000	160	A0	240		N/P	
10100001	161	A1	241		s	
10100010	162	A2	242		t	
10100011	163	A3	243		u	
10100100	164	A4	244		v	
10100101	165	A5	245		w	
10100110	166	A6	246		x	
10100111	167	A7	247		y	
10101000	168	A8	250		z	
10101001	169	A9	251		N/P	
10101010	170	AA	252		N/P	
10101011	171	AB	253		N/P	
10101100	172	AC	254		N/P	
10101101	173	AD	255		N/P	
10101110	174	AE	256		N/P	
10101111	175	AF	257		N/P	
10110000	176	B0	260		N/P	
10110001	177	B1	261		N/P	
10110010	178	B2	262		N/P	
10110011	179	B3	263		N/P	
10110100	180	B4	264		N/P	
10110101	181	B5	265		N/P	
10110110	182	B6	266		N/P	
10110111	183	B7	267		N/P	
10111000	184	B8	270		N/P	
10111001	185	B9	271		N/P	
10111010	186	BA	272		N/P	
10111011	187	BB	273		N/P	
10111100	188	BC	274		N/P	
10111101	189	BD	275		N/P	
10111110	190	BE	276		N/P	
10111111	191	BF	277		N/P	

Binary	Decimal	Hex	Octal	ASCII	EBCDIC	BCD
11000000	192	C0	300		{	
11000001	193	C1	301		A	
11000010	194	C2	302		B	
11000011	195	C3	303		C	
11000100	196	C4	304		D	
11000101	197	C5	305		E	
11000110	198	C6	306		F	
11000111	199	C7	307		G	
11001000	200	C8	310		H	
11001001	201	C9	311		I	
11001010	202	CA	312		N/P	
11001011	203	CB	313		N/P	
11001100	204	CC	314			
11001101	205	CD	315		N/P	
11001110	206	CE	316			
11001111	207	CF	317		N/P	
11010000	208	D0	320		}	
11010001	209	D1	321		J	
11010010	210	D2	322		K	
11010011	211	D3	323		L	
11010100	212	D4	324		M	
11010101	213	D5	325		N	
11010110	214	D6	326		O	
11010111	215	D7	327		P	
11011000	216	D8	330		Q	
11011001	217	D9	331		R	
11011010	218	DA	332		N/P	
11011011	219	DB	333		N/P	
11011100	220	DC	334		N/P	
11011101	221	DD	335		N/P	
11011110	222	DE	336		N/P	
11011111	223	DF	337		N/P	
11100000	224	E0	340		\	
11100001	225	E1	341		N/P	
11100010	226	E2	342		S	
11100011	227	E3	343		T	
11100100	228	E4	344		U	
11100101	229	E5	345		V	
11100110	230	E6	346		W	
11100111	231	E7	347		X	
11101000	232	E8	350		Y	
11101001	233	E9	351		Z	
11101010	234	EA	352		N/P	
11101011	235	EB	353		N/P	
11101100	236	EC	354			
11101101	237	ED	355		N/P	
11101110	238	EE	356		N/P	
11101111	239	EF	357		N/P	

Binary	Decimal	Hex	Octal	ASCII	EBCDIC	BCD
11110000	240	F0	360		0	
11110001	241	F1	361		1	
11110010	242	F2	362		2	
11110011	243	F3	363		3	
11110100	244	F4	364		4	
11110101	245	F5	365		5	
11110110	246	F6	366		6	
11110111	247	F7	367		7	
11111000	248	F8	370		8	
11111001	249	F9	371		9	
11111010	250	FA	372			
11111011	251	FB	373		N/P	
11111100	252	FC	374		N/P	
11111101	253	FD	375		N/P	
11111110	254	FE	376		N/P	
11111111	255	FF	377		EO	

Special Characters are as follows:

ACK	Acknowledge
BEL	Bell
BS	Backspace
BYP	Bypass
CAN	Cancel
CC	Cursor Control
CR	Carriage Return
CU1	Customer Use 1
CU2	Customer Use 2
CU3	Customer Use 3
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
DEL	Delete
DLE	Data Link Escape
DS	Digit Select
EM	End of Medium
ENQ	Enquiry
EO	Eight Ones
EOT	End of Transmission
ESC	Escape
ETB	End of Transmission Block
ETX	End of Text
FF	Form Feed
FS	Field Separator (EBCDIC)
FS	File Separator (ASCII)
GE	Graphics Escape
GS	Group Separator
HT	Horizontal Tab
IFS	Interchange File Separator
IGS	Interchange Group Separator
IL	Idle
IRS	Interchange Record Separator
IUS	Interchange Unit Separator
LC	Lower Case
LF	Line Feed
NAK	Negative Acknowledge
NL	New Line
NUL	Null
PF	Punch Off
PN	Punch On
RES	Restore
RLF	Reverse Line Feed
RS	Reader Stop (EBCDIC)
RS	Record Separator (ASCII)
SI	Shift In
SM	Set Mode
SMM	Start of Manual Message
SO	Shift Out
SOH	Start of Heading
SOS	Start of Significance
SP	Space
STX	Start of Text
SUB	Substitute
SYN	Synchronous Idle
TM	Tape Mark
UC	Upper Case
US	Unit Separator
VT	Vertical Tab

The relationships between the order in which characters appear in a code, as well as the number of characters that appear, have many subtle effects on the ability to translate data from one code to another.

As computers normally compare data based upon the binary value of the characters, the "collating sequence" in which the computer determines the order of data for sorting or comparison purposes differs between these three character sets. With ASCII and BCD characters, numbers occur before letters and therefore have a "lower" value, while numbers have a "higher" value in EBCDIC. In ASCII, lower case letters have a higher value than upper case letters, while the reverse is true in EBCDIC. If one were to read a tape written in order in EBCDIC and translate the information from EBCDIC into corresponding ASCII characters, the records may no longer appear to be in order.

Consider this example. You have received a tape containing personalized license plate numbers, presorted by license plate number in EBCDIC order, for processing on an ASCII computer. Your software translates EBCDIC "A"s into ASCII "A"s, "B"s into "B"s, "9"s into "9"s, etc. Because the tape was sorted in EBCDIC, the record containing "NITEOWL" comes before the record containing "NI4NI", as "T" comes before "4" in EBCDIC. When your COBOL or BASIC program looks for the record "NI4NI" and encounters "NITEOWL", it assumes that the "NI4NI" record does not exist, because in ASCII "NITEOWL" comes after "NI4NI".

In addition to these order abberations, not all characters are translatable. There is no EBCDIC equivalent of ASCII's square brackets, "[" and "]", while there is no ASCII equivalent of EBCDIC's cent sign "¢". Neither ASCII or EBCDIC define BCD's not equal to sign "≠".

In some character codes, such as ASCII, where the number of bits in the character set does not equal the number of bits recorded, the remaining bit(s) may be all set to 0 or 1 by convention, or in some cases, ignored. If not compensated for, this may become a source of incompatibility. For example, some Control Data computers always record the most significant bit as a 1, while other computers do not correctly interpret the ASCII data unless the most significant bit is set to 0.

Numeric Data Representations

In addition to these incompatibilities, there are other difficulties in interpreting data written by one machine when read by another. For some computer languages, in particular FORTRAN but also COBOL to a lesser extent, it is common to store numeric data in an "internal", or binary format. This method of storing information is more efficient from both a time and space standpoint. Internal format numbers are usually stored in a very compact fashion, requiring a minimum of storage space. They also require no interpretation to convert them from display characters to the binary form used internally by the computer.

As long as internal number representations are being read on a machine of the same type as that on which they were created, they are efficient time savers. When data is transferred by tape from one machine to another with different computers handling numbers in different ways, attempting to convert internal representations of numbers from one machine to another can be quite difficult.

Binary Numbers

Binary numbers (integers as discussed here) are frequently used to store prefixes containing record and block counts for variable length records. In addition, binary repetition counters occur in variable length COBOL records.

There are three principal areas in which binary numbers vary. First, the length of the binary number varies from computer to computer, with most computers able to handle more than one length of binary numbers. Second, the order in which bytes occur when representing binary numbers varies. And last of all, the method of representing signed numbers varies.

Binary numbers are frequently written to tape in lengths of 6, 8, 12, 15, 16, 18, 24, 30, 32, 36, 48, 60 and 64 bits. (Other lengths may also occur). Dealing with various byte lengths is not difficult for most programmers, as words can usually be extended or truncated to the nearest suitable length.

There are two common orders in which binary numbers are stored. Most mini and microcomputers store a binary number as a series of bytes with the least significant byte first, while most large computers store binary numbers with the most significant byte first. Although each of these conventions has its merits, the most significant implication to the 1/2" tape user is that bytes may have to be swapped around when reading a binary number from a tape created on a computer using a protocol differing from that of the computer reading the tape.

The broad variance in methods of storing negative binary numbers pose a considerable challenge in interpreting the numbers when transferred from one computer to another. Some computers store negative numbers in the same fashion as they store positive numbers, except that one usually high order bit is used as a sign bit. Much more common is the practice of storing binary numbers as "complements", with "two's complement" notation being more common on more recent computers and "one's complement" appearing on older machines. In addition to these methods, hybrids occur.

If you remember back to the days when adding machines could only add, you probably remember that you could subtract by taking the subtrahend, complementing each digit with 9, adding this to the minuend, and adding 1. The 9's complement was formed by replacing each digit with with the digit derived by subtracting from 9, i.e., 8 for 1, 7 for 2, 6 for 3, etc. It was necessary to supply as many leading zeros (which became leading nines when complemented) as there were digits on the adding machine, and the result aways carried (ringing the bell on the old machines), unless the subtrahend was larger than the minuend. This same principle is used in binary as one's complement notation. A one's complement negative number is represented as the complement of the positive number, with 1s replacing 0s and 0s replacing 1. But as with the adding machines, when you add a simple complement serving as a negative number, you have to add a one to the sum to come up with the final result.

Back to the adding machine analogy. To perform division, it was necessary to perform repetitious subtractions. Although the repeat key of the adding machine allowed this, it wouldn't repeat the process of adding the complement and then adding the one. Therefore, you learned that you could add one to the nine's complement of a number and have a number that could be directly added to yield the same result as a subtraction. This number, one more than the nine's complement, became known as the "ten's complement". (Philosophers cringe at this nomenclature). The same principle (and same misnomer) apply to binary arithmetic. The "two's complement" representation of a binary number is one more than the one's complement of the same number.

With both the two's complement and the one's complement representations, high order bits beyond the significance of a negative number are represented as binary 1s. When transferring signed binary numbers from from one computer to another where the length of the binary number is to be changed, high order 1 bits must be used to fill out negative one's and two's complement numbers, while 0s are used to fill out positive numbers.

A common hybrid form of storing negative numbers represents numbers as biased numbers. Normally the number stored is the sum of the number to be represented and a fixed biasing value. This biasing value is usually a power of two, and the bit represented by this power of two effectively constitutes a sign bit, or more accurately, the complement of the sign bit. The remaining bits of the number then become the actual value (for positive numbers) and the one's or two's complement of the value for negative numbers, depending upon the convention used.

Whenever binary numbers are transferred between dissimilar computers, it is necessary to consider the length, order and signing convention when transfering binary numbers from one computer to another to correctly compute their intended values.

Decimal Numbers

ANSI COBOL recognizes a COMPUTATIONAL-3 data type, otherwise called "packed decimal", which frequently appears on tape. This is a true, binary coded, decimal data format that utilizes the "upper" (or most significant) "nibble" (half byte, or four bits) of a data byte to store the more significant decimal digit and the "lower" nibble to store the less significant digit, combining two digits into each byte of decimal data. Most large computer hardware is able to add and subtract this format of data directly. These decimal numbers are stored with the most significant byte first, and contain an odd number of decimal digits. The last byte of a decimal number contains a sign designator appearing in the less significant nibble. This contains a hexadecimal "C" for positive numbers and a hexadecimal "D" for negative numbers.

Another common decimal format used on tape is the "zoned" decimal format. This format consists of the EBCDIC (or possibly ASCII) values for each decimal digit, using one byte per digit. The least significant decimal digit, however, is modified on negative numbers to be the character that would be generated by punching that digit on a Hollerith card together with a zone punch of "-". This yields the characters "}" through "R" replacing "0" through "9", respectively, as the least significant digits of negative decimal numbers.

Floating Point Numbers

The method of storing internal floating point numbers varies greatly in representation from one machine to the next. Only recently has some degree of standardization taken hold. Three of the more common formats will be discussed here.

The IBM 360 (and up) series of computers store floating point numbers as hexadecimal floating point representations, with the sign of the mantissa stored as the first, or leftmost bit. The characteristic, referred to as the exponent, is a hexadecimal (base sixteen) exponent consisting of the next seven bits in excess-64 notation, with exponent values of 0 through 127 representing exponents of -64 through 63, respectively. The mantissa, referred to as the fraction, is normalized to four bits yielding a hexadecimal point notation, with the most significant hexadecimal digit, or nibble, occuring in the leftmost position. The fraction and exponent are computed such that the fraction represents a number between 1/16th and 1, which is then raised by the exponent to a power of 16. Floating point numbers occur in precisions of 4, 8 and 16 bytes. (Note that this is a hexadecimal point notation, not a hexadecimal representation of a binary point notation. The characteristic expresses a power of 16, not a power of two or a power of ten, and the mantissa is normalized to a significant hexadecimal digit).

The IBM Personal Computer and other such machines that use formats compatible with the Intel 8087 Numeric Data Processor use a binary floating point format. (This format complies with the Institute of Electrical and Electronics Engineers, IEEE, proposed standard 754 for Binary Floating Point Arithmetic). In this representation the leftmost bit contains the sign of the mantissa with the next eight or twelve bits containing the characteristic, referred to as the biased exponent. As with the previously described format, the characteristic is biased, but by 127 for the 32 bit format (with an eight bit characteristic) and by 1023 for the 64 bit format with a twelve bit characteristic. The mantissa, referred to as the significand, is normalized, or shifted, so that the most significant bit of the significand is assumed to be a one and not stored. The remaining bits contain the stored part of the significand with the more significant bits stored first. The significand and the binary exponent are computed such that the binary point appears between the assumed one and the stored significand, yielding a number with a value between one and two raised to a binary power. A value of zero is represented by all fields containing zeros.

Control Data's large computers (6000, 7600, and Cyber 70 series) also store data as binary floating point numbers. As with the other formats, the first bit contains the sign of the mantissa. The characteristic is referred to also as the biased exponent, and is biased by 1024 (2000 octal), with negative exponents expressed in one's complement notation. The remaining 48 bits of the 60 bit number contains the mantissa, referred to as the "integer coefficient". This format assumes that the binary point follows the 48 bits of the integer coefficient. The number 1 would be expressed as an integer coefficient of 2^{+47} with a biased exponent of 976. This represents 2 raised to the 47th power times the binary exponent, 2^{-47}, yielding a one. (The 976 is one less than 1024 minus 47, using the one's complement rules for a negative number).

In summary, we have reviewed data representations consisting of character, binary, decimal and floating point data, with characters represented in standard codes such as ASCII, EBCDIC, BCD and others. 1/2" tape is suitable for storing the images of most computer memories, which, in addition to these above described formats, can store information in an unlimited variety of useful representations. When transferring data, data representations must match the application.

Labeling Standards

Most tapes used by large computer systems for their own purposes (as opposed to the interchange of data with other systems) are labeled in a fashion conforming to the operating system standard for that system. In most cases, these operating system standards conform to one of the various levels defined by the American National Standards Institute (ANSI, formerly USASI) for labeling magnetic tapes. Due to the many variations and interpretations of the meanings of data within the labels, differences between these operating systems as well as code set differences between machines, it is not normally possible to interchange tapes between two unlike operating systems, even though both purport to label tapes in accordance with ANSI standards.

The ANSI standards for unlabeled interchange tapes, with data followed by two file marks for a single file tape, or files separated with file marks and two file marks after the last file on the tape for multi-file tapes, normally provide the simplest, most interchangable method of data transfer.

Various levels of standard labeled tapes are defined. The most complex format supports multiple files per reel of tape, as well as multiple reels of tape per file. The other formats can generally be considered to be subsets of this format.

Each label record consists of an 80 byte block of tape data. Labels are separated from data records by file marks, with additional file marks separating ending label records for one file from header records for the next file.

The first four bytes of a label record indicate the type of label record. These types are specified by three alphabetic characters and one numeric character. They may be in EBCDIC (which is most common) or ASCII, which is also permitted.

A standard labeled tape begins with a VOL1 record. This record specifies the six character volume serial number, identifying the tape. Additional information, depending upon system, may indicate the type of tape, such as scratch, backup, system, etc.

The VOL1 label is followed by the HDR1 record for the first file on the tape. This label contains file identification information such as file (or dataset) name, creation date, expiration date, generation number, sequence number, etc. Dates are normally stored as Julian dates, with the first two characters containing the year number and the remaining three digits containing the day of the year. The Block Count and Record Count fields, bytes 55-60 and 61 through 67, are filled with numeric (decimal) zeros for header labels. These fields contain decimal block and record counts for data files on EOV (end of volume) and EOF (end of file) label records.

The HDR2 label record, if included, normally contains information critical to data I/O, such as block length and record length (positions 6-10 and 11-15), record format, code type, equipment type, variable length offset information, etc.

Other HDRn label records may also appear.

After the file header record, user records may appear. The information stored in these records is selected by the user, most commonly by a COBOL program, as COBOL supports user labels. UHR1 through UHR9 are permissible user records.

A file mark preceeds the actual data file. Record and block counts for the purpose of EOF and EOV records include only records and blocks between the file marks delimiting data files.

After the file mark indicating the end of a data file are the ending labels. One EOF label is written for each HDR record, with the EOF1 record containing the same information as the HDR1 (except for the updated block and record count), the EOF2 the same as the HDR2, etc. UTL (user trailing labels) label records are written similarly to correspond with UHL label records. A file mark follows the ending labels.

A HDR1 label record for a subsequent file may follow the file mark after the ending labels, or a second file mark may be written to indicate the logical end of tape. If another file follows, the block and record counts are reset to zero and start over for the next file.

If the physical end of tape is encountered with more data to be written, the data file is interrupted with a file mark, which is followed by an EOV1 label record. The EOV1 label record is of the same format as the EOF1 record, containing the current block and record counts for all blocks in the data file preceeding the end of volume. The data file continues on the next reel after the VOL1, HDR1 and other label records have been repeated with updated values for the block and record counters.

The multireel multifile "tape" ends, as do single reel tapes, with a double file mark.

Translation

Due to the preponderance of both ASCII and EBCDIC tapes (as well as computer systems), translation becomes a common operation when processing 1/2" tapes.

In addition to the problem discussed earlier concerning different apparent collating sequences for data encoded using different code sets, other considerations are necessary when translating data from one code set to another.

Translation may be performed through several different mechanisms when reading or writing tape data. The tape drive or formatter may itself perform translation, usually in hardware, when translation is selected by the interface. This is usually the fastest way to translate data.

Translation may also be performed by the system software, such as the operating system. This method allows the translation to be transparent to the user's application program, providing, as with hardware translation, all data is to be translated. System software data translation is usually quite efficient as it can take advantage of the ability many computers have to translate data with a specialized machine instruction. Use of this type of instruction is normally not provided to programs written in higher level languages.

Finally, translation may be performed by the application program. This provides the greatest flexibility, but is also the slowest and most cumbersome.

Translation of all data may not be useful when parts of the data are encoded in a character set (such as ASCII or EBCDIC) while other parts remain in binary, decimal or floating point. This requires selective translation of only the character data within records, which is usually not possible with system software or hardware. Translated binary and packed decimal data is usually meaningless.

Because ASCII contains characters not appearing in EBCDIC, and vice versa, all characters cannot be generally translated. Most translation tables make some assumptions for these characters. It goes without saying that characters that are not common to both character sets participating in a translation will not be correctly translated for all applications. Also, unless the translation tables are specifically modified for such a purpose, translation, for example, from ASCII to EBCDIC and back to ASCII will not necessarily yield the original text.

Translation is usually performed, whether in hardware, software or applications program, by taking the binary value of the data fetched from tape and using it as an index, or offset, into a translation table. Thus, the first value appearing in a translation table contains the value that replaces the first character in the character set to be translated. The secondsecond value is the translation of the second character in the character set, and so on.

Sample translation tables, used to translate ASCII to EBCDIC and EBCDIC to ASCII follow on the next page. The ASCII to EBCDIC translation used translates FS to IFS, GS to IGS, RS to IRS and US to IUS. The carrot, ^, does not appear in EBCDIC, and is translated into a logical or, . DC3, normally used to turn a punch or printer on, is translated to PN (Punch On). DEL is translated by this table into EO (Eight Ones), instead of DEL in EBCDIC, as all one bits usually has a special meaning. Finally, the ASCII square brackets, [and], are translated into EBCDIC's printing characters Hook and Chair, making the translation reversable so that ASCII text may be translated to EBCDIC and then back, yielding the same result.

The EBCDIC to ASCII translation table shown reverses these exceptions. In this particular table, no two different EBCDIC characters are translated into the same ASCII character. The N/P indicator is used to designate non-printing characters. NONE is used to indicate cases where no ASCII equivalent is provided.

It is important to note that these tables may not be suitable for all translation applications. Different printer print chains, communications protocols, language interpreters, etc., may require different translation tables.

Also note that because ASCII is a seven bit code normally stored as eight bits in most computers, the high order bit must be ignored (zeroed) for translation.

ASCII to EBCDIC Translation

ASCII (HEX)	EBCDIC (HEX)	ASCII char	EBCDIC char	ASCII (HEX)	EBCDIC (HEX)	ASCII char	EBCDIC char
00	00	NUL	NUL	20	40	SP	SP
01	01	SOH	SOH	21	5A	!	!
02	02	STX	STX	22	7F	"	"
03	03	ETX	ETX	23	7B	#	#
04	37	EOT	EOT	24	5B	$	$
05	2D	ENQ	ENQ	25	6C	%	%
06	2E	ACK	ACK	26	50	&	&
07	2F	BEL	BEL	27	7D	'	'
08	16	BS	BS	28	4D	(	(
09	05	HT	HT	29	5D	)	)
0A	25	LF	LF	2A	5C	*	*
0B	0B	VT	VT	2B	4E	+	+
0C	0C	FF	FF	2C	6B	,	,
0D	0D	CR	CR	2D	60	-	-
0E	0E	SO	SO	2E	4B	.	.
0F	0F	SI	SI	2F	61	/	/
10	10	DLE	DLE	30	F0	0	0
11	11	DC1	DC1	31	F1	1	1
12	12	DC2	DC2	32	F2	2	2
13	34	DC3	PN	33	F3	3	3
14	3C	DC4	DC4	34	F4	4	4
15	3D	NAK	NAK	35	F5	5	5
16	32	SYN	SYN	36	F6	6	6
17	26	ETB	ETB	37	F7	7	7
18	18	CAN	CAN	38	F8	8	8
19	19	EM	EM	39	F9	9	9
1A	3F	SUB	SUB	3A	7A	:	:
1B	27	ESC	ESC	3B	5E	;	;
1C	1C	FS	IFS	3C	4C	<	<
1D	1D	GS	IGS	3D	7E	=	=
1E	1E	RS	IRS	3E	6E	>	>
1F	1F	US	IUS	3F	6F	?	?

Appendix A

ASCII to EBCDIC Translation

ASCII (HEX)	EBCDIC (HEX)	ASCII char	EBCDIC char	ASCII (HEX)	EBCDIC (HEX)	ASCII char	EBCDIC char
40	7C	@	@	70	97	p	p
41	C1	A	A	71	98	q	q
42	C2	B	B	72	99	r	r
43	C3	C	C	73	A2	s	s
44	C4	D	D	74	A3	t	t
45	C5	E	E	75	A4	u	u
46	C6	F	F	76	A5	v	v
47	C7	G	G	77	A6	w	w
48	C8	H	H	78	A7	x	x
49	C9	I	I	79	A8	y	y
4A	D1	J	J	7A	A9	z	z
4B	D2	K	K	7B	C0	{	{
4C	D3	L	L	7C	6A	|	
4D	D4	M	M	7D	D0	}	}
4E	D5	N	N	7E	A1	~	~
4F	D6	O	O	7F	07	DEL	DEL
50	D7	P	P				
51	D8	Q	Q				
52	D9	R	R				
53	E2	S	S				
54	E3	T	T				
55	E4	U	U				
56	E5	V	V				
57	E6	W	W				
58	E7	X	X				
59	E8	Y	Y				
5A	E9	Z	Z				
5B	CC	[					
5C	E0	\	\				
5D	CE	]					
5E	4F	^					
5F	6D	_	_				
60	79	`	`				
61	81	a	a				
62	82	b	b				
63	83	c	c				
64	84	d	d				
65	85	e	e				
66	86	f	f				
67	87	g	g				
68	88	h	h				
69	89	i	i				
6A	91	j	j				
6B	92	k	k				
6C	93	l	l				
6D	94	m	m				
6E	95	n	n				
6F	96	o	o				

EBCDIC to ASCII Translation

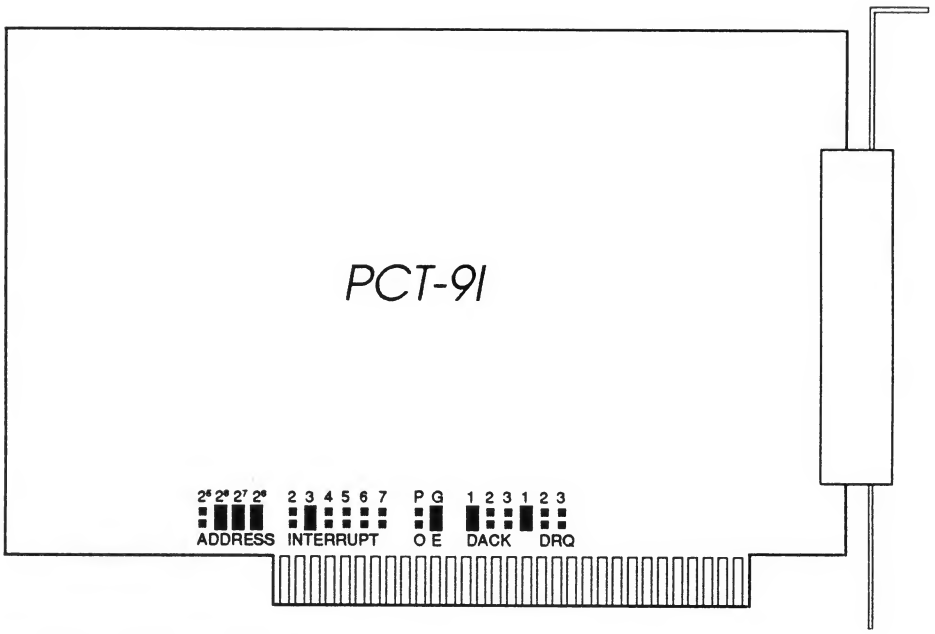
EBCDIC (HEX)	ASCII (HEX)	EBCDIC char	ASCII char	EBCDIC (HEX)	ASCII (HEX)	EBCDIC char	ASCII char
00	00	NUL	NUL	30	NONE	N/P	
01	01	SOH	SOH	31	NONE	N/P	
02	02	STX	STX	32	16	SYN	SYN
03	03	ETX	ETX	33	NONE	N/P	
04	NONE	PF		34	13	PN	DC3
05	09	HT	HT	35	NONE	RS	
06	NONE	LC		36	NONE	UC	
07	7F	DEL	DEL	37	04	EOT	EOT
08	NONE	GE		38	NONE	N/P	
09	NONE	RLF		39	NONE	N/P	
0A	NONE	SMM		3A	NONE	N/P	
0B	0B	VT	VT	3B	NONE	CU3	
0C	0C	FF	FF	3C	14	DC4	DC4
0D	0D	CR	CR	3D	15	NAK	NAK
0E	0E	SO	SO	3E	NONE	N/P	
0F	0F	SI	SI	3F	1A	SUB	SUB
10	10	DLE	DLE	40	20	SP	SP
11	11	DC1	DC1	41	NONE	N/P	
12	12	DC2	DC2	42	NONE	N/P	
13	NONE	TM		43	NONE	N/P	
14	NONE	RES		44	NONE	N/P	
15	NONE	NL		45	NONE	N/P	
16	08	BS	BS	46	NONE	N/P	
17	NONE	IL		47	NONE	N/P	
18	18	CAN	CAN	48	NONE	N/P	
19	19	EM	EM	49	NONE	N/P	
1A	NONE	CC		4A	NONE	¢	
1B	NONE	CU1		4B	2E	.	.
1C	1C	IFS	FS	4C	3C	<	<
1D	1D	IGS	GS	4D	28	(	(
1E	1E	IRS	RS	4E	2B	+	+
1F	1F	IUS	US	4F	NONE		
20	NONE	DS		50	26	&	&
21	NONE	SOS		51	NONE	N/P	
22	NONE	FS		52	NONE	N/P	
23	NONE	N/P		53	NONE	N/P	
24	NONE	BYP		54	NONE	N/P	
25	0A	LF	LF	55	NONE	N/P	
26	17	ETB	ETB	56	NONE	N/P	
27	1B	ESC	ESC	57	NONE	N/P	
28	NONE	N/P		58	NONE	N/P	
29	NONE	N/P		59	NONE	N/P	
2A	NONE	SM		5A	21	!	!
2B	NONE	CU2		5B	24	$	$
2C	NONE	N/P		5C	2A	*	*
2D	05	ENQ	ENQ	5D	29	)	)
2E	06	ACK	ACK	5E	3B	;	;
2F	07	BEL	BEL	5F	NONE		

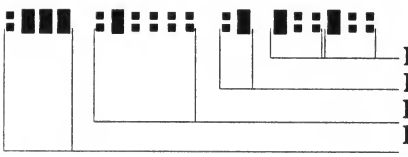
EBCDIC to ASCII Translation

EBCDIC (HEX)	ASCII (HEX)	EBCDIC char	ASCII char	EBCDIC (HEX)	ASCII (HEX)	EBCDIC char	ASCII char
60	2D	-	-	90	NONE	N/P	
61	2F	/	/	91	6A	j	j
62	NONE	N/P		92	6B	k	k
63	NONE	N/P		93	6C	l	l
64	NONE	N/P		94	6D	m	m
65	NONE	N/P		95	6E	n	n
66	NONE	N/P		96	6F	o	o
67	NONE	N/P		97	70	p	p
68	NONE	N/P		98	71	q	q
69	NONE	N/P		99	72	r	r
6A	NONE			9A	NONE	N/P	
6B	2C	,	,	9B	NONE	N/P	
6C	25	%	%	9C	NONE	N/P	
6D	5F	_	_	9D	NONE	N/P	
6E	3E	>	>	9E	NONE	N/P	
6F	3F	?	?	9F	NONE	N/P	
70	NONE	N/P		A0	NONE	N/P	
71	NONE	N/P		A1	7E	~	~
72	NONE	N/P		A2	73	s	s
73	NONE	N/P		A3	74	t	t
74	NONE	N/P		A4	75	u	u
75	NONE	N/P		A5	76	v	v
76	NONE	N/P		A6	77	w	w
77	NONE	N/P		A7	78	x	x
78	NONE	N/P		A8	79	y	y
79	60	`	`	A9	7A	z	z
7A	3A	:	:	AA	NONE	N/P	
7B	23	#	#	AB	NONE	N/P	
7C	40	@	@	AC	NONE	N/P	
7D	27	'	'	AD	NONE	N/P	
7E	3D	=	=	AE	NONE	N/P	
7F	22	"	"	AF	NONE	N/P	
80	NONE	N/P		B0	NONE	N/P	
81	61	a	a	B1	NONE	N/P	
82	62	b	b	B2	NONE	N/P	
83	63	c	c	B3	NONE	N/P	
84	64	d	d	B4	NONE	N/P	
85	65	e	e	B5	NONE	N/P	
86	66	f	f	B6	NONE	N/P	
87	67	g	g	B7	NONE	N/P	
88	68	h	h	B8	NONE	N/P	
89	69	i	i	B9	NONE	N/P	
8A	NONE	N/P		BA	NONE	N/P	
8B	NONE	N/P		BB	NONE	N/P	
8C	NONE	N/P		BC	NONE	N/P	
8D	NONE	N/P		BD	NONE	N/P	
8E	NONE	N/P		BE	NONE	N/P	
8F	NONE	N/P		BF	NONE	N/P	

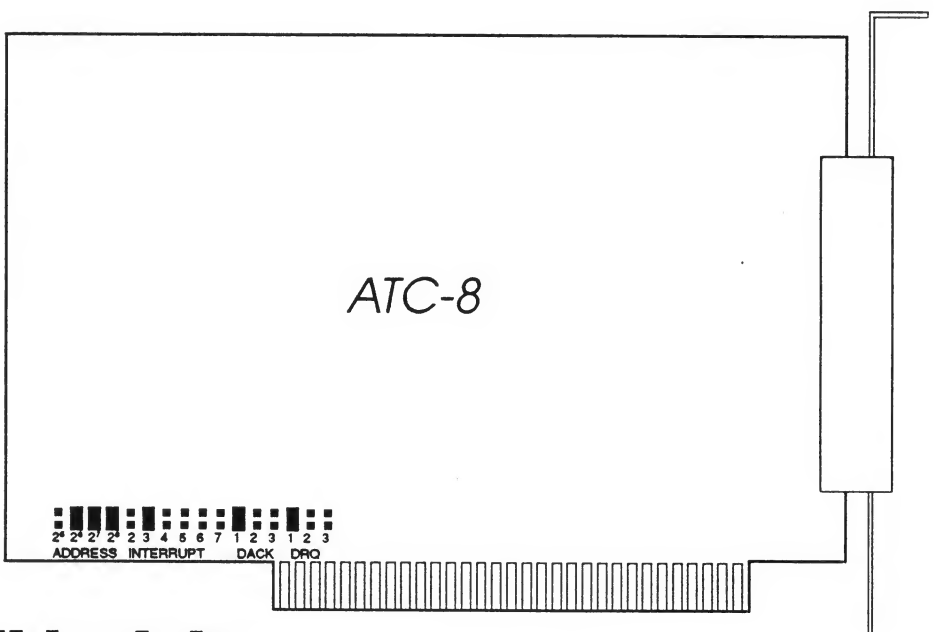
EBCDIC to ASCII Translation

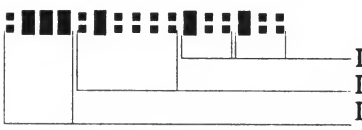
EBCDIC (HEX)	ASCII (HEX)	EBCDIC char	ASCII char	EBCDIC (HEX)	ASCII (HEX)	EBCDIC char	ASCII char
C0	7B	{	{	F0	30	0	0
C1	41	A	A	F1	31	1	1
C2	42	B	B	F2	32	2	2
C3	43	C	C	F3	33	3	3
C4	44	D	D	F4	34	4	4
C5	45	E	E	F5	35	5	5
C6	46	F	F	F6	36	6	6
C7	47	G	G	F7	37	7	7
C8	48	H	H	F8	38	8	8
C9	49	I	I	F9	39	9	9
CA	NONE	N/P		FA	NONE		
CB	NONE	N/P		FB	NONE	N/P	
CC	5B		[	FC	NONE	N/P	
CD	NONE	N/P		FD	NONE	N/P	
CE	5D		]	FE	NONE	N/P	
CF	NONE	N/P		FF	NONE	EO	
D0	7D	}	}				
D1	4A	J	J				
D2	4B	K	K				
D3	4C	L	L				
D4	4D	M	M				
D5	4E	N	N				
D6	4F	O	O				
D7	50	P	P				
D8	51	Q	Q				
D9	52	R	R				
DA	NONE	N/P					
DB	NONE	N/P					
DC	NONE	N/P					
DD	NONE	N/P					
DE	NONE	N/P					
DF	NONE	N/P					
E0	5C	\	\				
E1	NONE	N/P					
E2	53	S	S				
E3	54	T	T				
E4	55	U	U				
E5	56	V	V				
E6	57	W	W				
E7	58	X	X				
E8	59	Y	Y				
E9	5A	Z	Z				
EA	NONE	N/P					
EB	NONE	N/P					
EC	NONE						
ED	NONE	N/P					
EE	NONE	N/P					
EF	NONE	N/P					



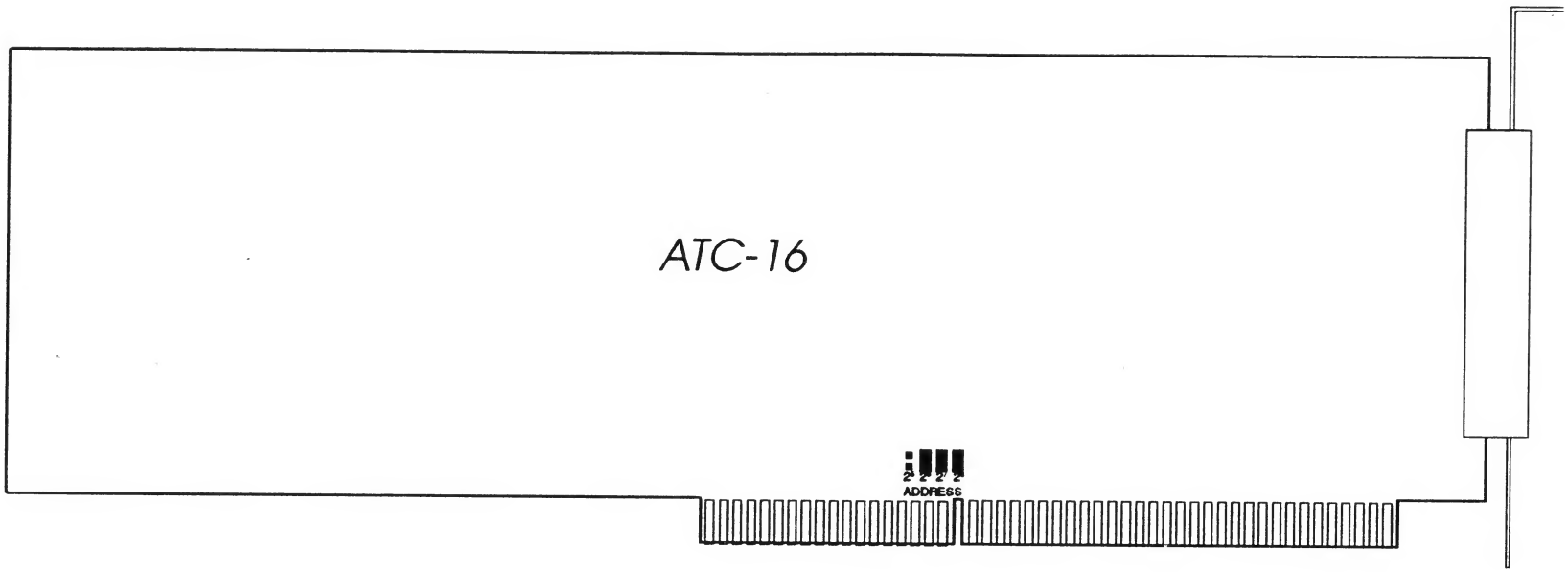


DMA Acknowledge/DMA Request (Both must be the same)
PO/GE (Should stay on GE)
Interrupt Level
I/O Address (See Chart B-1)





DMA Acknowledge/DMA Request (Both must be the same)
Interrupt Level
I/O Address (See Chart B-1)



I/O Address

(See Chart B-1)

CHART B-1

0200H	0280H	0300H	0380H
0220H	02A0H	0320H	03A0H
0240H	02C0H	0340H	03C0H
0260H	02E0H	0360H	03E0H

Jumpers and Addresses in TINSTALL must match for proper operation.
If you are using Interrupt Level 0, the jumper settings for Interrupt Level don't matter.
If you are using DMA 0 or 4, the jumper settings for DRQ and DACK don't matter.